CS 211: Intro to Computer Architecture

4.2: C Preprocessing and Compilation

Minesh Patel

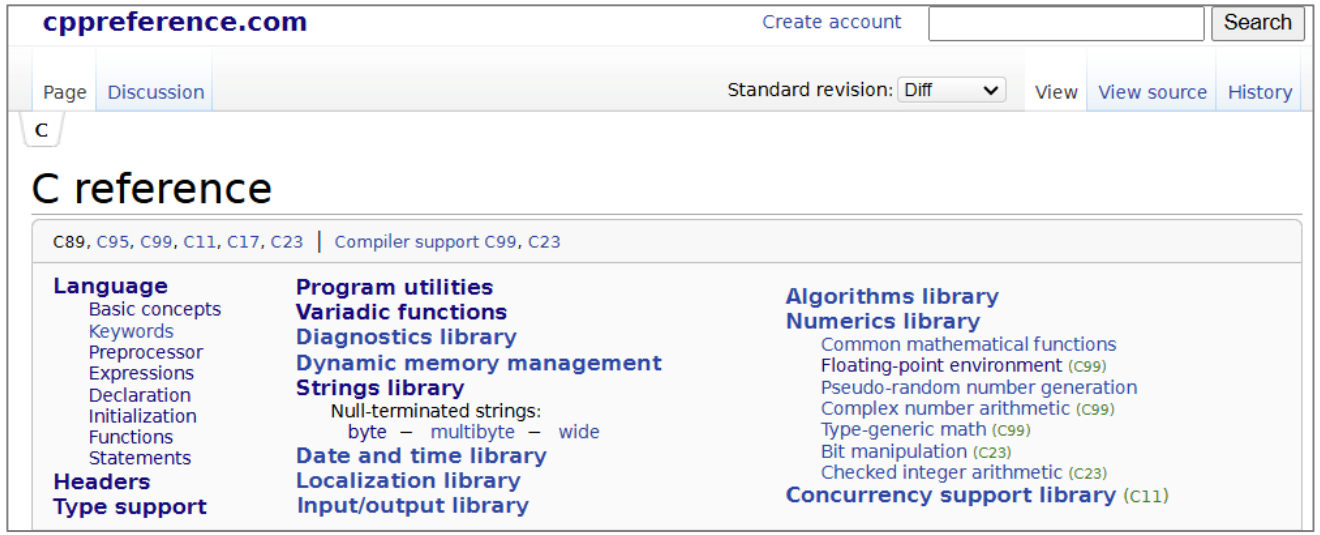
Spring 2025 – Thursday 13 February

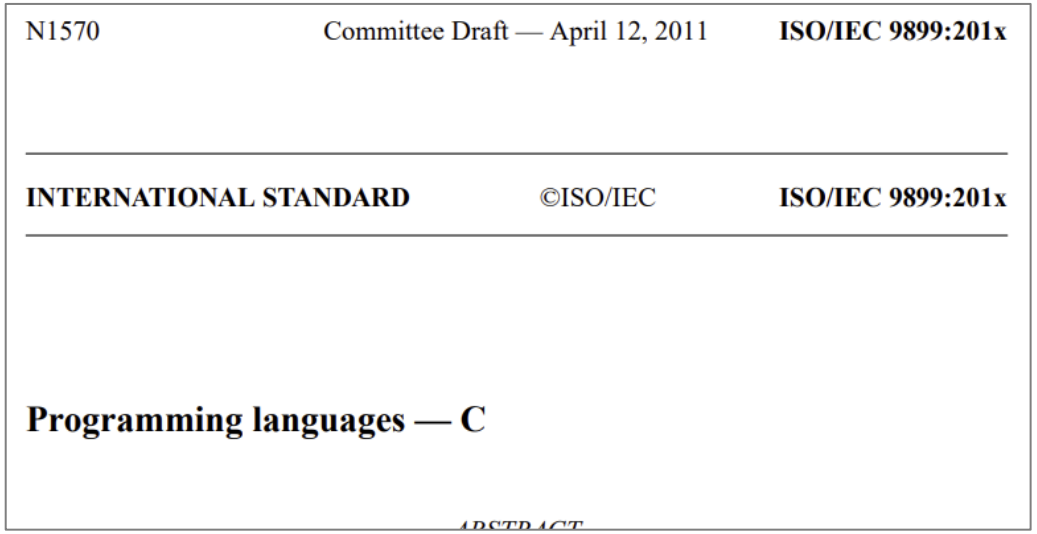
Announcements

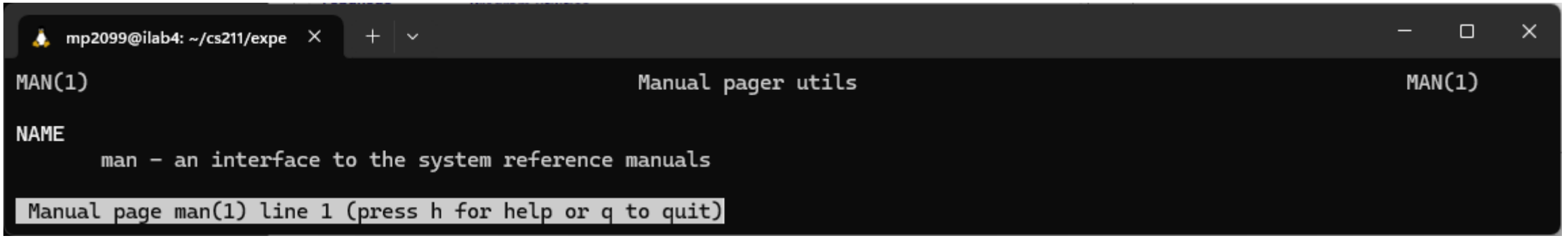
- **PA2:** Released yesterday, due end of **next week** (Sunday, Feb 23 @ 23:59)
  - Submit on Gradescope via your GitHub Classroom repository
- **WA2** due at the end of **this week** (Sunday, Feb 16 @ 23:59)

Recommended Reference Material

- I will refer to several references extensively in the coming lectures







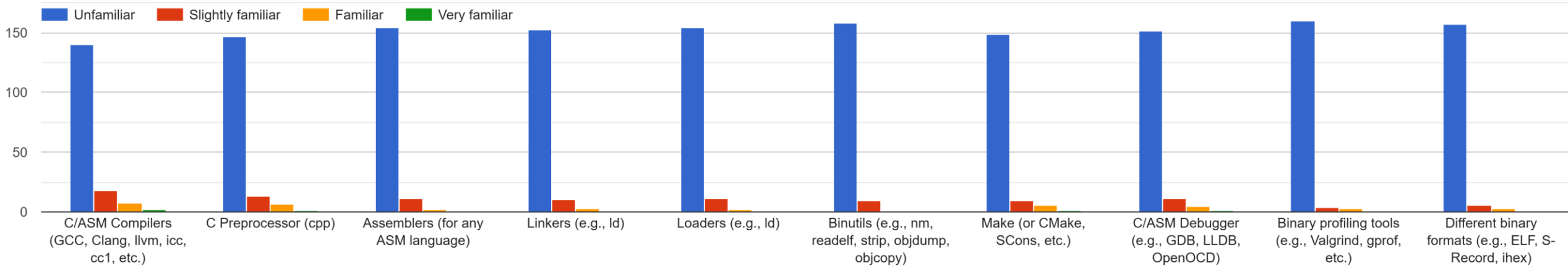
Shout-out to Related Material

- "C for Java Programmers" guides
  - George Ferguson (University of Rochester), *"C for Java Programmers,"* 2016.
  - Jason Maassen (VU Amsterdam), *"C for Java Programmers."*
  - Tomasz Muldner (Acadia University), *"C for Java Programmers,"* 2000.
  - Charlie McDowell (UC Santa Cruz), *"C for Java Programmers,"* 2000.
  - And many more…
- Other versions of CS211
  - Profs. Gavva, Menendez, Huang, Nagarakatte, Nath (+ others before my time)
- Some related courses with public materials
  - UC Berkeley's CS 61C
  - CMU's CS213
  - Harvard's CS50

C-Related Topics Survey

- 168 / 210 responses (80%)

[C/ASM Toolchain] Are you familiar with:



Agenda

- **Platforms and Compilers**
- "Hello World" Analysis
- Compiling C Code

Executable Program Representations

- Programs have **different binary representations** on each platform

x86_64

amd
linux

ilab

https://dlcdnwebimgs.asus.com/gain/d47ac467-7878-47b7-a29a-c9adfb80b9bd/

intel
windows

My laptop

https://www.ultrabookreview.com/64501-asus-vivobook-pro-16x-oled-review/

aarch64

qualcomm
android

Smartphones

https://m.media-amazon.com/images/I/61eQRrP5STL._AC_SL1000_.jpg

apple
ios

https://www.apple.com/newsroom/2020/06/apple-reimagines-the-iphone-experience-with-ios-14/

powerpc

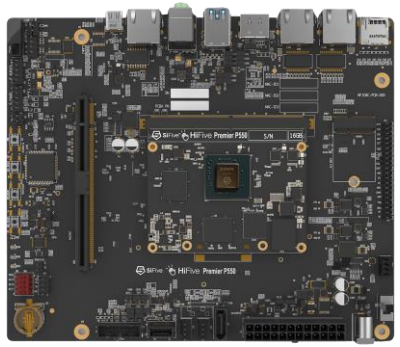
ibm
Xbox OS

Xbox 360

https://m.media-amazon.com/images/I/81+lz2g6bJL._SL1500_.jpg

riscv64

sifive
linux



Single-Board Computer

https://www.sifive.com/boards

Myriad Platforms

```
enum ArchType {
    UnknownArch,

    arm,            // ARM (little endian): arm, armv.*, xscale
    armeb,          // ARM (big endian): armeb
    aarch64,        // AArch64 (little endian): aarch64
    aarch64_be,     // AArch64 (big endian): aarch64_be
    aarch64_32,     // AArch64 (little endian) ILP32: aarch64_32
    arc,            // ARC: Synopsys ARC
    avr,            // AVR: Atmel AVR microcontroller
    bpfel,          // eBPF or extended BPF or 64-bit BPF (little endian)
    bpfeb,          // eBPF or extended BPF or 64-bit BPF (big endian)
    csky,           // CSKY: csky
    dxil,           // DXIL 32-bit DirectX bytecode
    hexagon,        // Hexagon: hexagon
    loongarch32,    // LoongArch (32-bit): loongarch32
    loongarch64,    // LoongArch (64-bit): loongarch64
    m68k,           // M68k: Motorola 680x0 family
    mips,           // MIPS: mips, mipsallegrex, mipsr6
    mipsel,         // MIPSEL: mipsel, mipsallegrexe, mipsr6el
    mips64,         // MIPS64: mips64, mips64r6, mipsn32, mipsn32r6
    mips64el,       // MIPS64EL: mips64el, mips64r6el, mipsn32el, mipsn32r6el
    msp430,         // MSP430: msp430
    ppc,            // PPC: powerpc
    ppcle,          // PPCLE: powerpc (little endian)
    ppc64,          // PPC64: powerpc64, ppu
    ppc64le,        // PPC64LE: powerpc64le
    r600,           // R600: AMD GPUs HD2XXX - HD6XXX
    amdgcn,         // AMDGCN: AMD GCN GPUs
    riscv32,        // RISC-V (32-bit): riscv32
    riscv64,        // RISC-V (64-bit): riscv64
    sparc,          // Sparc: sparc
    sparcv9,        // Sparcv9: Sparcv9
    sparcel,        // Sparc: (endianness = little). NB: 'Sparcle' is a CPU variant
    systemz,        // SystemZ: s390x
    tce,            // TCE (http://tce.cs.tut.fi/): tce
    tcele,          // TCE little endian (http://tce.cs.tut.fi/): tcele
    thumb,          // Thumb (little endian): thumb, thumbv.*
    thumbeb,        // Thumb (big endian): thumbeb
    x86,            // X86: i[3-9]86
    x86_64,         // X86-64: amd64, x86_64
    xcore,          // XCore: xcore
    xtensa,         // Tensilica: Xtensa
    nvptx,          // NVPTX: 32-bit
    nvptx64,        // NVPTX: 64-bit
    amdil,          // AMDIL
    amdil64,        // AMDIL with 64-bit pointers
    hsail,          // AMD HSAIL
    hsail64,        // AMD HSAIL with 64-bit pointers
    spir,           // SPIR: standard portable IR for OpenCL 32-bit version
    spir64,         // SPIR: standard portable IR for OpenCL 64-bit version
    spirv,          // SPIR-V with logical memory layout.
    spirv32,        // SPIR-V with 32-bit pointers
    spirv64,        // SPIR-V with 64-bit pointers
    kalimba,        // Kalimba: generic kalimba
    shave,          // SHAVE: Movidius vector VLIW processors
    lanai,          // Lanai: Lanai 32-bit
    wasm32,         // WebAssembly with 32-bit pointers
    wasm64,         // WebAssembly with 64-bit pointers
    renderscript32, // 32-bit RenderScript
    renderscript64, // 64-bit RenderScript
    ve,             // NEC SX-Aurora Vector Engine
    LastArchType = ve
  };
```

```
enum OSType {
    UnknownOS,

    Darwin,
    DragonFly,
    FreeBSD,
    Fuchsia,
    IOS,
    KFreeBSD,
    Linux,
    Lv2, // PS3
    MacOSX,
    NetBSD,
    OpenBSD,
    Solaris,
    UEFI,
    Win32,
    ZOS,
    Haiku,
    RTEMS,
    NaCl, // Native Client
    AIX,
    CUDA,   // NVIDIA CUDA
    NVCL,   // NVIDIA OpenCL
    AMDHSA, // AMD HSA Runtime
    PS4,
    PS5,
    ELFIAMCU,
    TvOS,      // Apple tvOS
    WatchOS,   // Apple watchOS
    BridgeOS,  // Apple bridgeOS
    DriverKit, // Apple DriverKit
    XROS,      // Apple XROS
    Mesa3D,
    AMDPAL,     // AMD PAL Runtime
    HermitCore, // HermitCore Unikernel/Multikernel
    Hurd,       // GNU/Hurd
    WASI,       // Experimental WebAssembly OS
    Emscripten,
    ShaderModel, // DirectX ShaderModel
    LiteOS,
    Serenity,
    Vulkan, // Vulkan SPIR-V
    LastOSType = Vulkan
 };

 enum VendorType {
     UnknownVendor,

     Apple,
     PC,
     SCEI,
     Freescale,
     IBM,
     ImaginationTechnologies,
     MipsTechnologies,
     NVIDIA,
     CSR,
     AMD,
     Mesa,
     SUSE,
     OpenEmbedded,
     Intel,
     LastVendorType = Intel
   };
```

```
 enum EnvironmentType {
    UnknownEnvironment,

    GNU,
    GNUT64,
    GNUABIN32,
    GNUABI64,
    GNUEABI,
    GNUEABIT64,
    GNUEABIHF,
    GNUEABIHFT64,
    GNUF32,
    GNUF64,
    GNUSF,
    GNUX32,
    GNUILP32,
    CODE16,
    EABI,
    EABIHF,
    Android,
    Musl,
    MuslABIN32,
    MuslABI64,
    MuslEABI,
    MuslEABIHF,
    MuslF32,
    MuslSF,
    MuslX32,
    LLVM,

    MSVC,
    Itanium,
    Cygnus,
    CoreCLR,
    Simulator, // Simulator variants of other systems,
e.g., Apple's iOS
    MacABI,    // Mac Catalyst variant of Apple's iOS
deployment target.

    // Shader Stages
    // The order of these values matters, and must be
kept in sync with the
    // language options enum in Clang. The ordering is
enforced in
    // static_asserts in Triple.cpp and in Clang.
    Pixel,
    Vertex,
    Geometry,
    Hull,
    Domain,
    Compute,
    Library,
    RayGeneration,
    Intersection,
    AnyHit,
    ClosestHit,
    Miss,
    Callable,
    Mesh,
    Amplification,
    OpenCL,
    OpenHOS,

    PAuthTest,

    LastEnvironmentType = PAuthTest
 };
```

https://llvm.org/doxygen/Triple_8h_source.html

Compiling to Machine Code

- The machine **only** understands executable binaries (i.e., machine code)
- All other languages must be **translated** to machine code

Human Readable

C Source Code

(Platform-Independent)

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

Machine Readable

Executable Binary (C Compiler)

(Platform-Dependent)

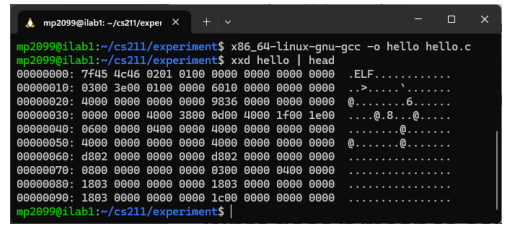
input

output

Machine Readable

Executable Binary (Your App)

(Platform-Dependent)



Sort-of Human Readable

Assembly Source Code

(Platform-Dependent)

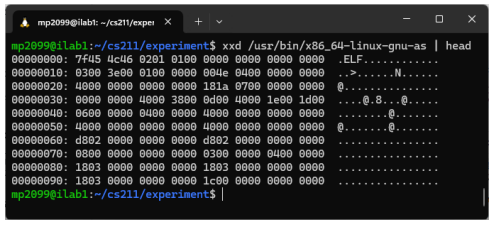
```
        .string "Hello World!"
main:
        addi    sp,sp,-32
        sd      ra,24(sp)
        sd      s0,16(sp)
        addi    s0,sp,32
        mv      a5,a0
        sd      a1,-32(s0)
        sw      a5,-20(s0)
        lla     a0,.LC0
        call    puts
        li      a5,0
        mv      a0,a5
        ld      ra,24(sp)
        ld      s0,16(sp)
        addi    sp,sp,32
        jr      ra
```

Machine Readable

Executable Binary (Assembler)

(Platform-Dependent)

input

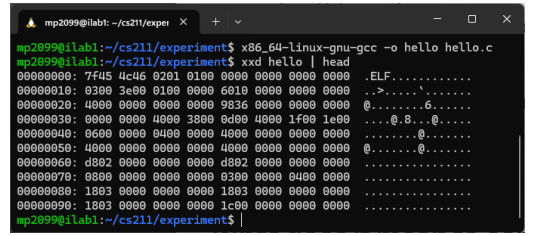


output

Machine Readable

Executable Binary (Your App)

(Platform-Dependent)



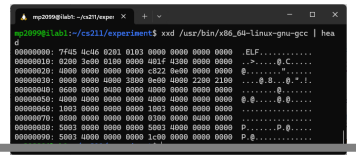
Compilers vs.... Other Ways to Run Code

C Source Code (Platform-Independent)

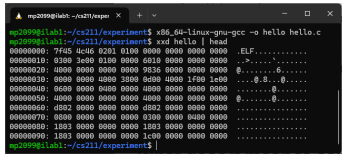
```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

Executable Binary (C Compiler) (Platform-Dependent)

input → output



Executable Binary (Your App) (Platform-Dependent)

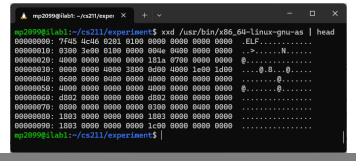


Assembly Source Code (Platform-Dependent)

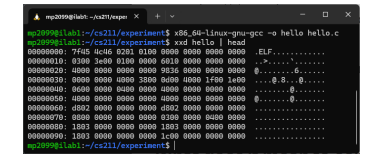
```
        .string "Hello World!"
main:
        addi    sp,sp,-32
        sd      ra,24(sp)
        sd      s0,16(sp)
        addi    s0,sp,32
        mv      a5,a0
        sd      a1,-32(s0)
        sw      a5,-20(s0)
        lla     a0,.LC0
        call    puts
        li      a5,0
        mv      a0,a5
        ld      ra,24(sp)
        ld      s0,16(sp)
        addi    sp,sp,32
        jr      ra
```

Executable Binary (Assembler) (Platform-Dependent)

input → output



Executable Binary (Your App) (Platform-Dependent)

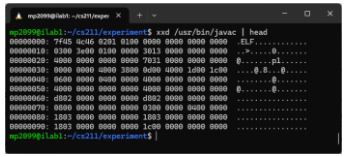


Java Source Code (Platform-Independent)

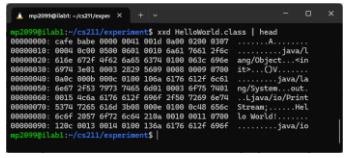
```
class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

Executable Binary (javac) (Platform-Dependent)

input → output

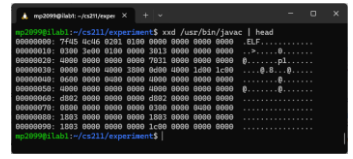


Java Bytecode (Platform-Independent)



Executable Binary (JVM) (Platform-Dependent)

input



Python Source Code (Platform-Independent)

```
import os

if __name__ == “__main__”:
        print(“Hello World!”)
        exit(os.EX_OK)
```

Executable Binary (Python Interpreter) (Platform-Dependent)

input

“Libraries” are Executable Code

Numpy Source Code
(Platform-Independent)

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

NumPy Source Code
(Platform-Dependent)

```
        .string "Hello World!"
main:
        addi    sp,sp,-32
        sd      ra,24(sp)
        sd      s0,16(sp)
        addi    s0,sp,32
        mv      a5,a0
        sd      a1,-32(s0)
        sw      a5,-20(s0)
        lla     a0,.LC0
        call    puts
        li      a5,0
        mv      a0,a5
        ld      ra,24(sp)
        ld      s0,16(sp)
        addi    sp,sp,32
        jr      ra
```

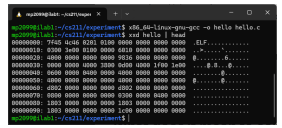
Python Source Code
(Platform-Independent)

```python
import numpy as np

if __name__ == “__main__”:
    print(np.array(1))
    exit(os.EX_OK)
```

Executable Binary (C Compiler)
(Platform-Dependent)

input → output

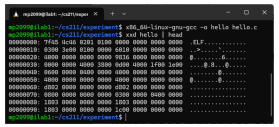
Executable Binary (NumPy Library)
(Platform-Dependent)



Executable Binary (Assembler)
(Platform-Dependent)

input → output

Executable Binary (NumPy Library)
(Platform-Dependent)



Executable Binary (Python Interpreter)
(Platform-Dependent)

input

Compiler “Targets”

Source Code (Platform-Independent)

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

Compiling on x86_64-linux

Executable Binary (C Compiler)
(Platform-Dependent)

input

```
mp2099@ilab1:~/cs211/experiment$ xxd /usr/bin/x86_64-linux-gnu-gcc | hea
d
00000000: 7f45 4c46 0201 0103 0000 0000 0000 0000  .ELF............
00000010: 0200 3e00 0100 0000 401f 4300 0000 0000  ..>.....@.C.....
00000020: 4000 0000 0000 0000 c822 0e00 0000 0000  @........".......
00000030: 0000 0000 4000 3800 0e00 4000 2200 2100  ....@.8...@.".!.
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 4000 0000 0000 4000 4000 0000 0000  @.@.....@.@.....
00000060: 1003 0000 0000 0000 1003 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 5003 0000 0000 0000 5003 4000 0000 0000  P.......P.@.....
00000090: 5003 4000 0000 0000 1c00 0000 0000 0000  P.@.............
mp2099@ilab1:~/cs211/experiment$
```

output

“native” compiler target: same platform as the host

“cross” compiler target: different platform from the host

Executable Binaries (Platform-Dependent)

Running on x86_64-linux

```
mp2099@ilab1:~/cs211/experiment$ x86_64-linux-gnu-gcc -o hello hello.c
mp2099@ilab1:~/cs211/experiment$ xxd hello | head
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 9836 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
mp2099@ilab1:~/cs211/experiment$
```

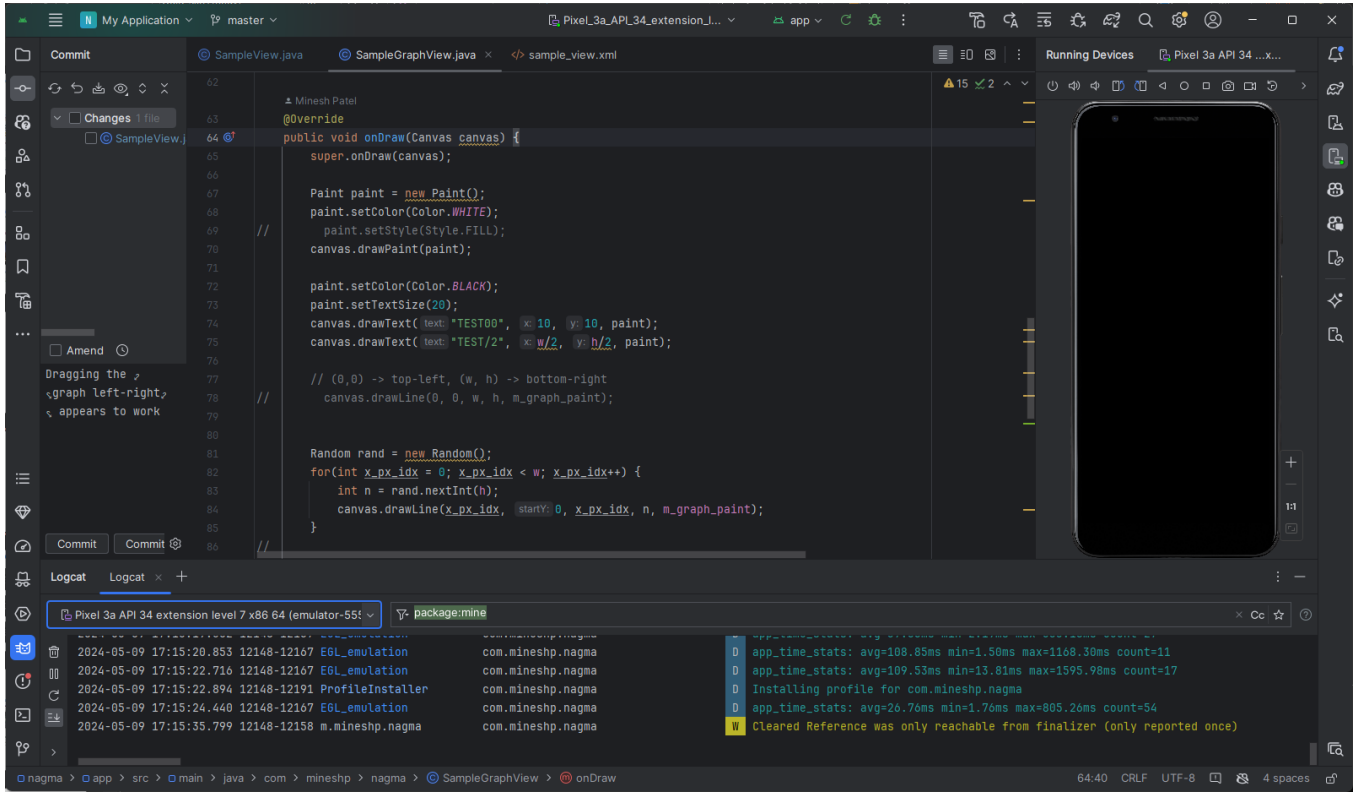
Running on riscv64-linux

```
mp2099@ilab1:~/cs211/experiment$ riscv64-linux-gnu-gcc -o hello hello.c
mp2099@ilab1:~/cs211/experiment$ xxd hello | head
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 f300 0100 0000 b005 0000 0000 0000  ................
00000020: 4000 0000 0000 0000 101a 0000 0000 0000  @...............
00000030: 0500 0000 4000 3800 0a00 4000 1c00 1b00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: 3002 0000 0000 0000 3002 0000 0000 0000  0.......0.......
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 7002 0000 0000 0000 7002 0000 0000 0000  p.......p.......
00000090: 7002 0000 0000 0000 2100 0000 0000 0000  p.......!.......
mp2099@ilab1:~/cs211/experiment$
```

Cross-Compilation Example

Android Studio

Compiling on `x86_64-windows`



Mobile App

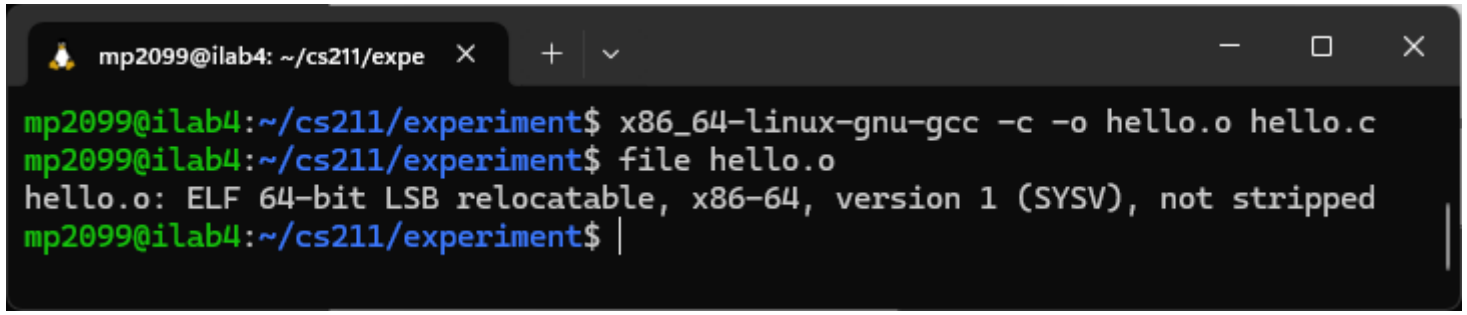
Running on `aarch64-android`

Smartphone

https://m.media-amazon.com/images/I/61eQRrP5STL._AC_SL1000_.jpg

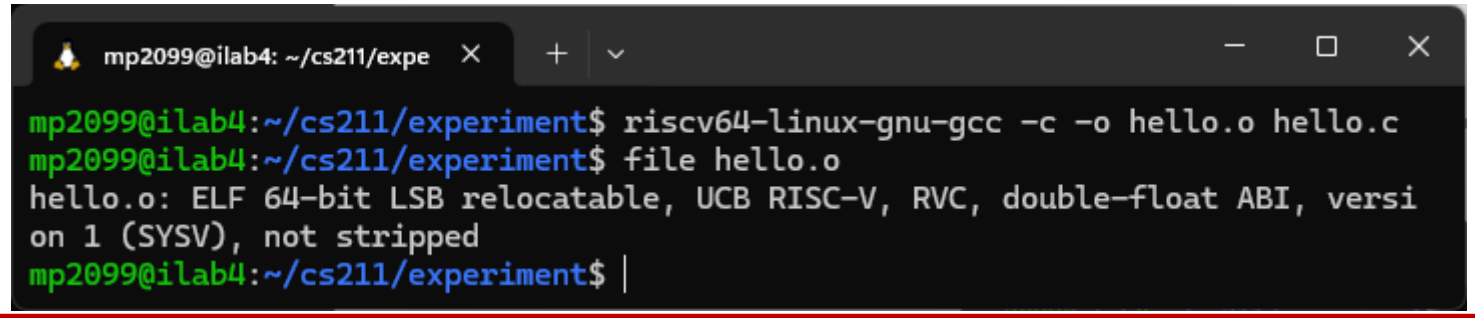
ilab: Three Different Compilers

Compiling on x86_64-linux

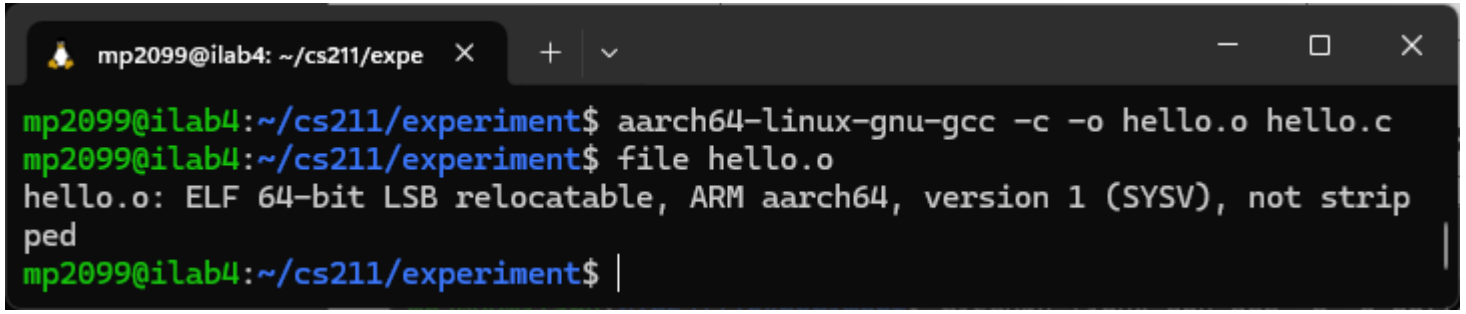


Targeting X86_64-linux

This Class



Targeting riscv64-linux



Targeting aarch64-linux

Agenda

- Platforms and Compilers
- **“Hello World” Analysis**
- Compiling C Code

C vs. Java "Hello World"

hello_world.c

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

Functions live alone in C files
(no classes or encapsulation)

HelloWorld.java

```java
class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

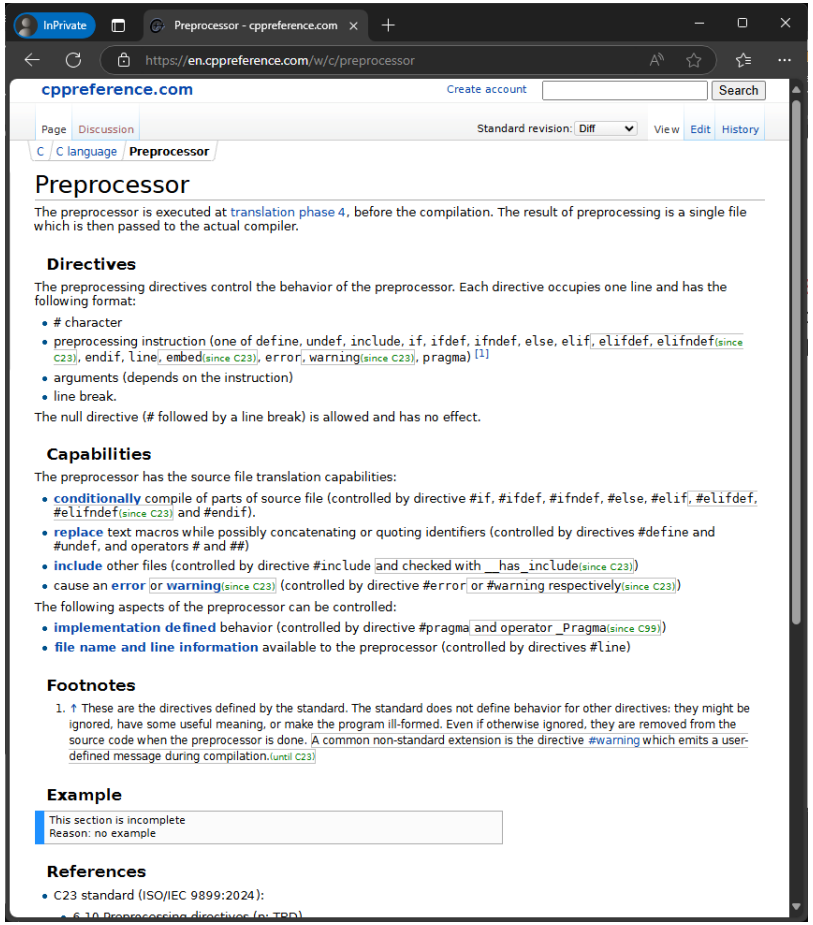
Preprocessor Directives

hello_world.c

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

"#include" is a preprocessor directive

- Tells the C compiler (specifically, the *preprocessor*) to copy/paste the entire contents of **stdlib.h** into this file
- That way, we can use functions/variables in **stdlib.h**



Preprocessor Directives

hello_world.c

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

stdlib.h Source Code

(yes, they use Git too!)

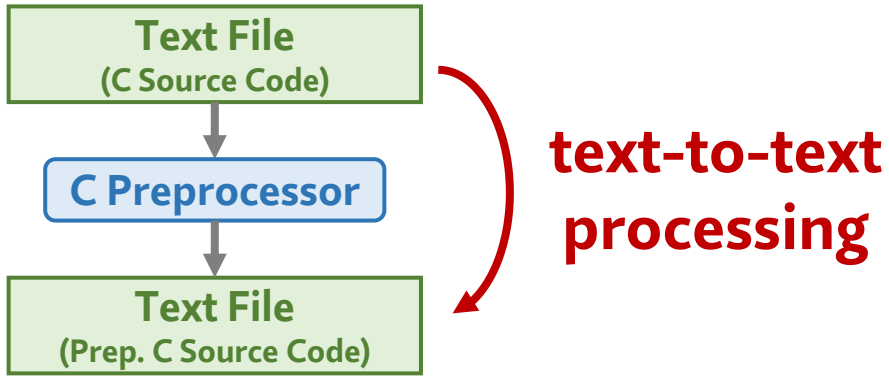
"#define" is another **preprocessor directive**

- Tells the *preprocessor* to replace all instances of **EXIT_FAILURE** with **1**
- Avoids magic numbers in the code!

```c
/* We define these the same for all machines.
   Changes from this to the outside world should be done in `_exit'. */
#define EXIT_FAILURE    1       /* Failing exit status. */
#define EXIT_SUCCESS    0       /* Successful exit status. */
```

preprocessor macro

C Preprocessing



hello_world.c

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

hello_world.c

```
mp2099@ilab1:~/cs211/experiment$ cat hello_world.c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
mp2099@ilab1:~/cs211/experiment$
```

hello_world.i

```
{
 struct _reent *_ptr;
 _ptr = _impure_ptr;
 return ((--(((_ptr)->_stdin))->_r < 0 ? __srget_r
(_ptr, ((_ptr)->_stdin)) : (int)(*(((_ptr)->_stdin
))->_p++)));
}
static __inline int
_putchar_unlocked(int _c)
{
 struct _reent *_ptr;
 _ptr = _impure_ptr;
 return (__sputc_r(_ptr, _c, ((_ptr)->_stdout)));
}

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return 0;
}
mp2099@ilab1:~/cs211/experiment$
```

C Preprocessing: Macro Bugs

- Be **very careful** with macros: text replacement **ignores** code correctness!

```
#define PI 3.14
#define TWO_PI PI + PI

float four_pi(void)
{
    return 2 * TWO_PI;
}
```

```
float four_pi(void)
{
    return 2 * 3.14 + 3.14;
}
```

```
#define PI 3.14
#define TWO_PI (PI + PI)

float four_pi(void)
{
    return 2 * TWO_PI;
}
```

```
float four_pi(void)
{
    return 2 * (3.14 + 3.14);
}
```

C Preprocessing: Function-Like Macro Bugs

- Be **very careful** with macros: text replacement **ignores** code correctness!

```
#define PI 3.14
#define PI_PLUS_N(x) PI + x

float four_pi(void)
{
    return 2 * PI_PLUS_N(PI);
}
```

```
float four_pi(void)
{
    return 2 * PI + PI;
}
```

```
#define PI 3.14
#define PI_PLUS_N(x) (PI + x)

float four_pi(void)
{
    return 2 * PI_PLUS_N(PI);
}
```

```
float four_pi(void)
{
    return 2 * (PI + PI);
}
```

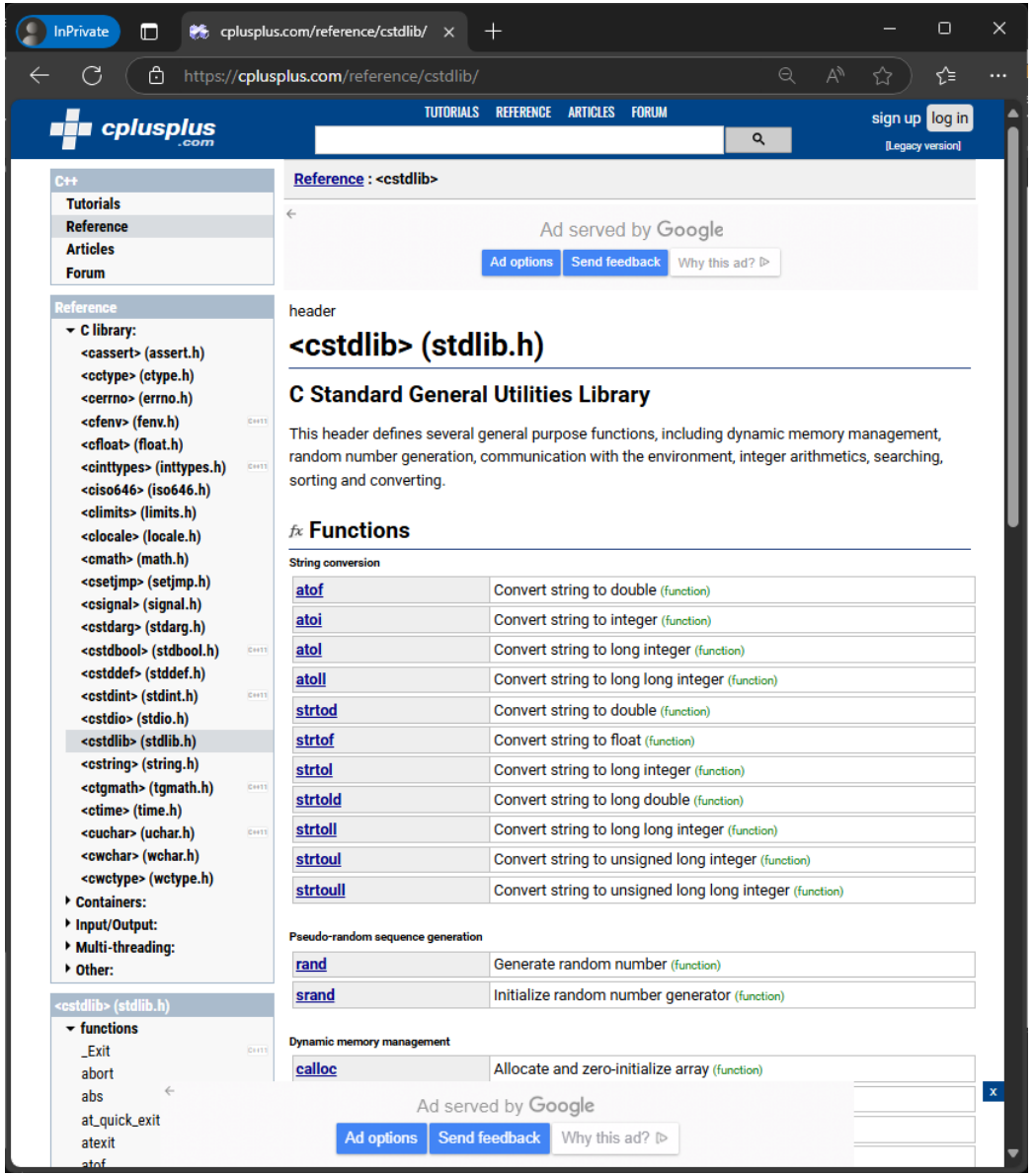
<stdlib.h>

hello_world.c

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

C Standard Library:

stdlib.h Documentation



<stdio.h>

hello_world.c

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

C Standard Library:

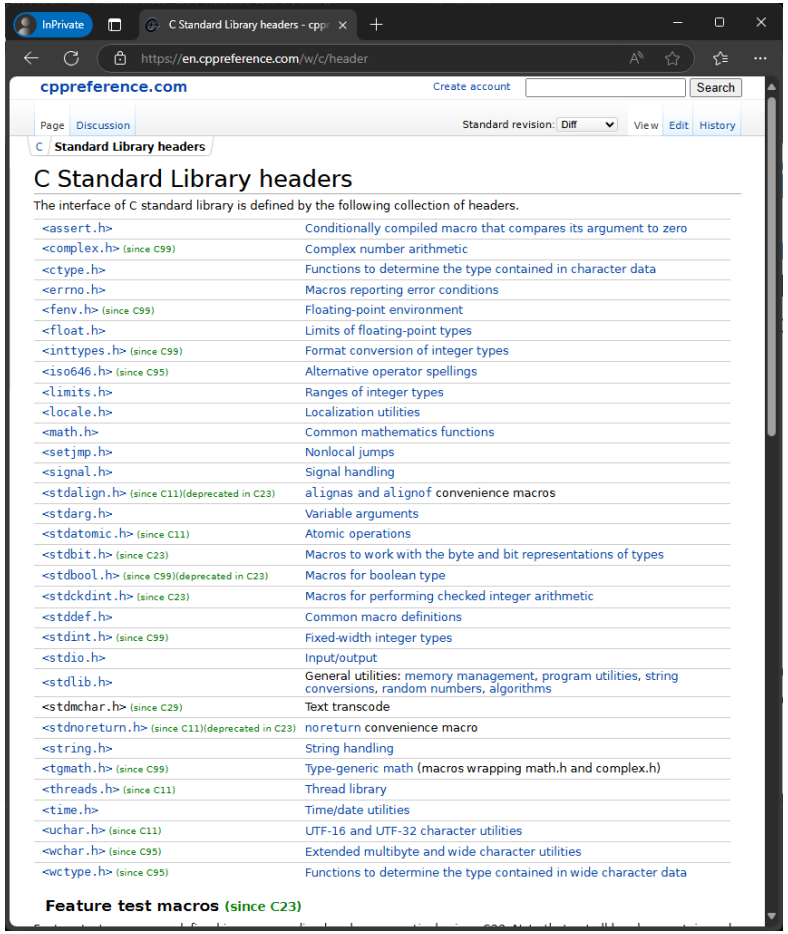
stdio.h Documentation

C Standard Libraries

- **C Standard Libraries** are… standardized
  - Basic utility functions and macros
  - Required by the C standard

hello_world.c

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```



- Anything else is yours to implement ☺
  - Can use “external libraries” (beyond this class)

C Starter Code: main

hello_world.c

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

- **Function** "`main`"
  - Where your program starts

```
mp2099@ilab1:~/cs211/experiment$ ./hello_world 1 two 0x3
```

- **Inputs** from the command line
  - `int argc`: number of program arguments
  - `char *argv[]`: array of args as ASCII strings
- **One output** to the command line
  - `int`: "return value" (here, **EXIT_SUCCESS**)

C Starter Code: main

- `int argc`: number of program arguments
- `char *argv[]`: list of arguments (ASCII strings)

hello_world.c

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

```
mp2099@ilab1:~/cs211/experiment$ ./hello_world 1 two 0x3
```

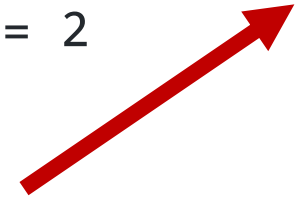
argc = 4

argv[0] = "./hello_world"
argv[1] = "1"
argv[2] = "two"
argv[3] = "0x3"

```
mp2099@ilab1:~/cs211/experiment$ ./hello_world .?-3/!
```

argc = 2

argv[0] = "./hello_world"
argv[1] = ".?-3/!"



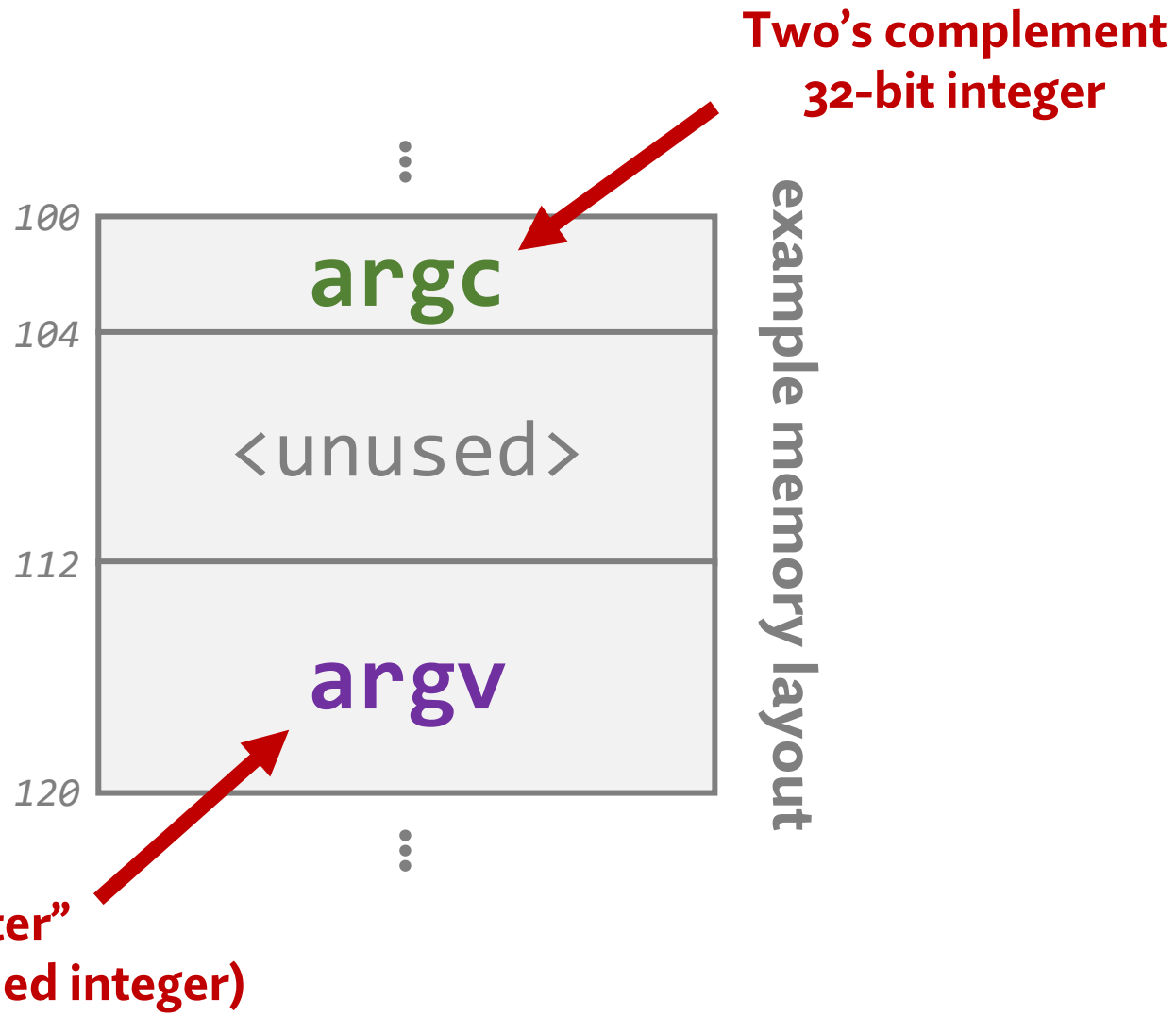
The CLI command is **usually** the first argument

C Starter Code: Objects in Memory

hello_world.c

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

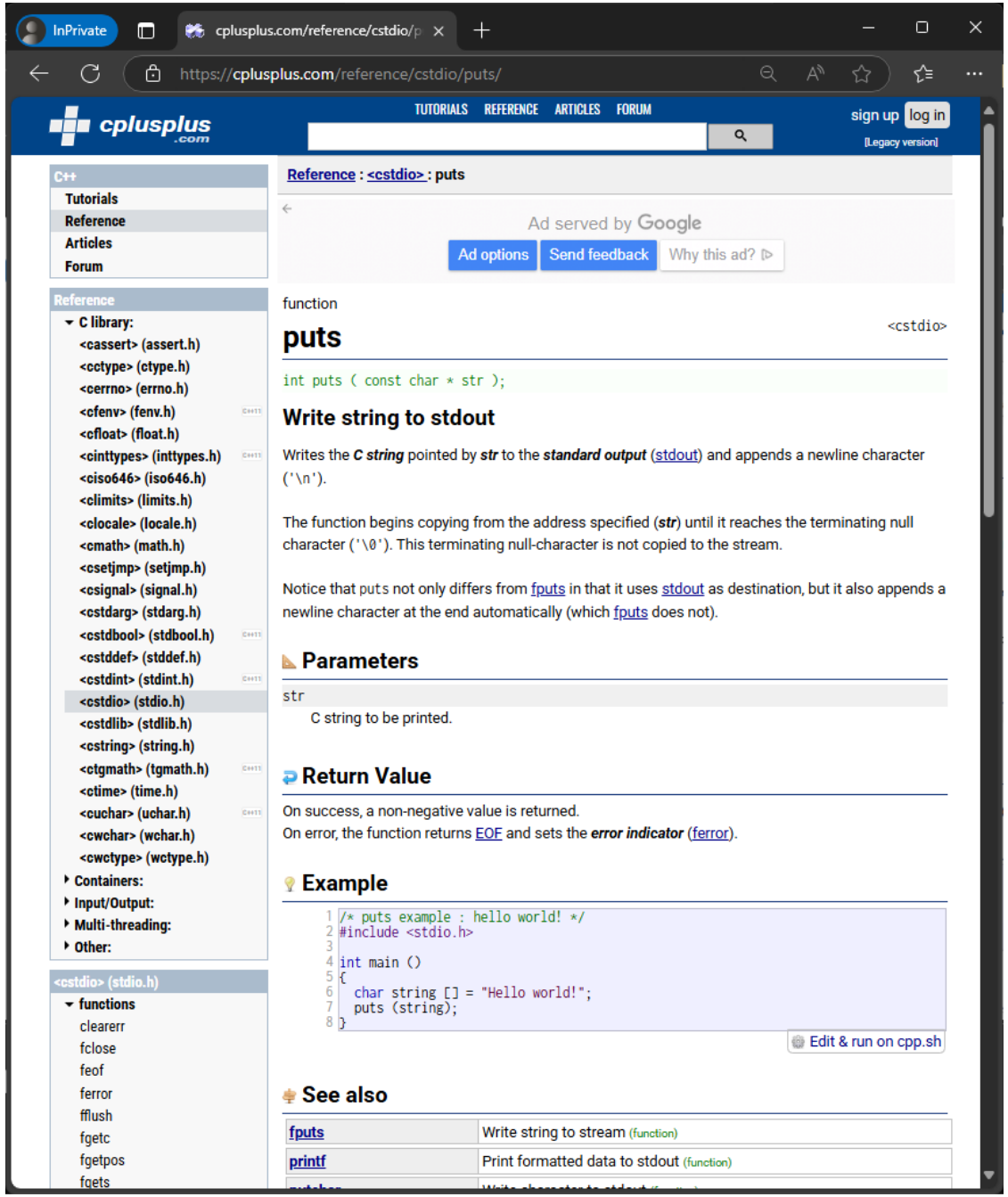
objects



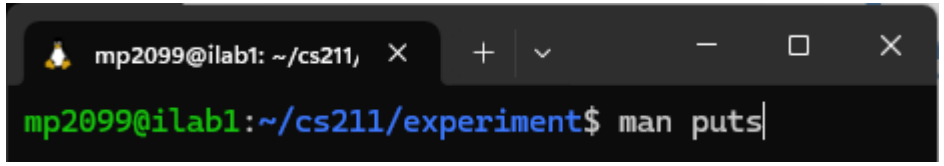
C Starter Code: puts

hello_world.c

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

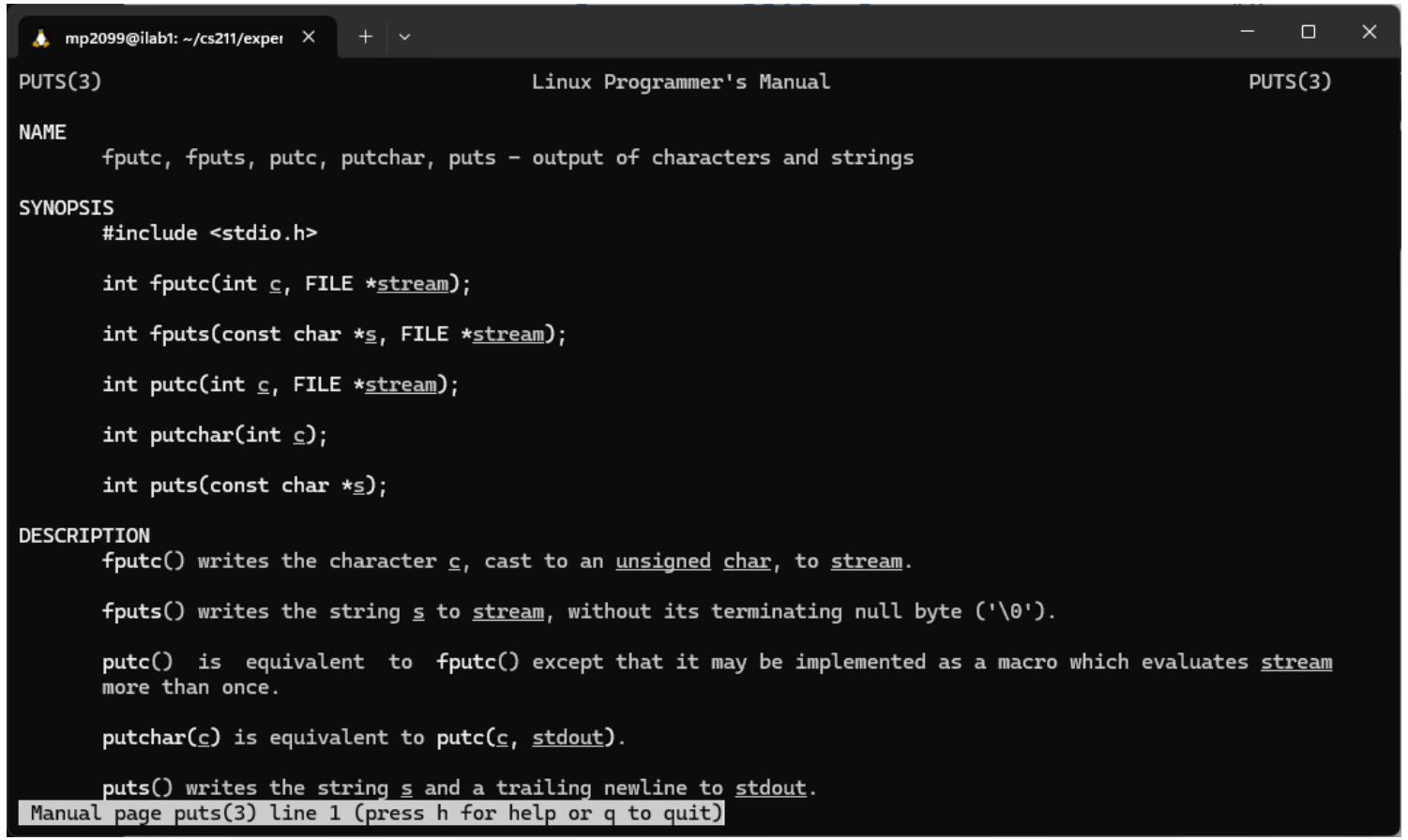


C Starter Code: puts



hello_world.c

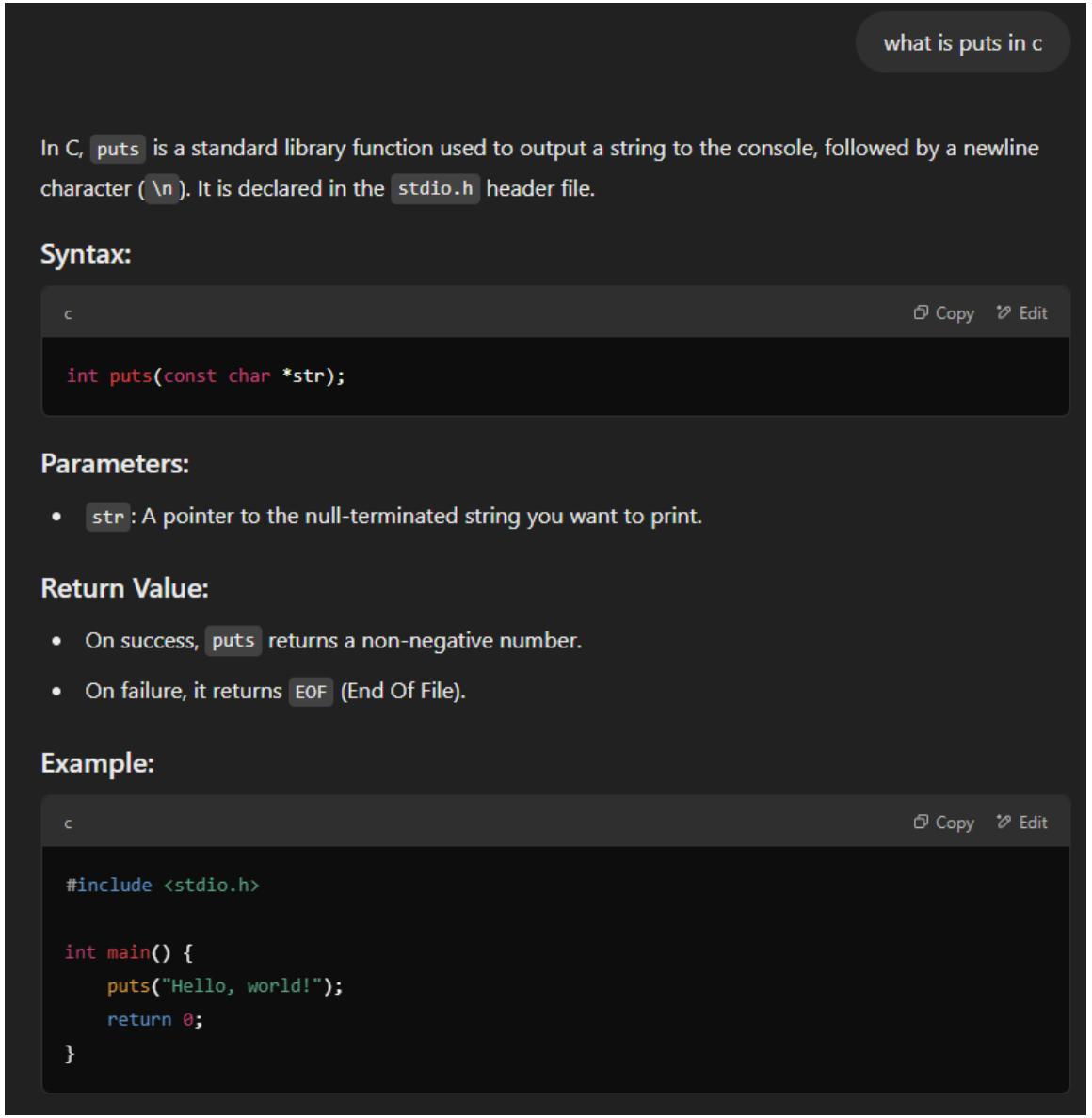
```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```



C Starter Code: puts

hello_world.c

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

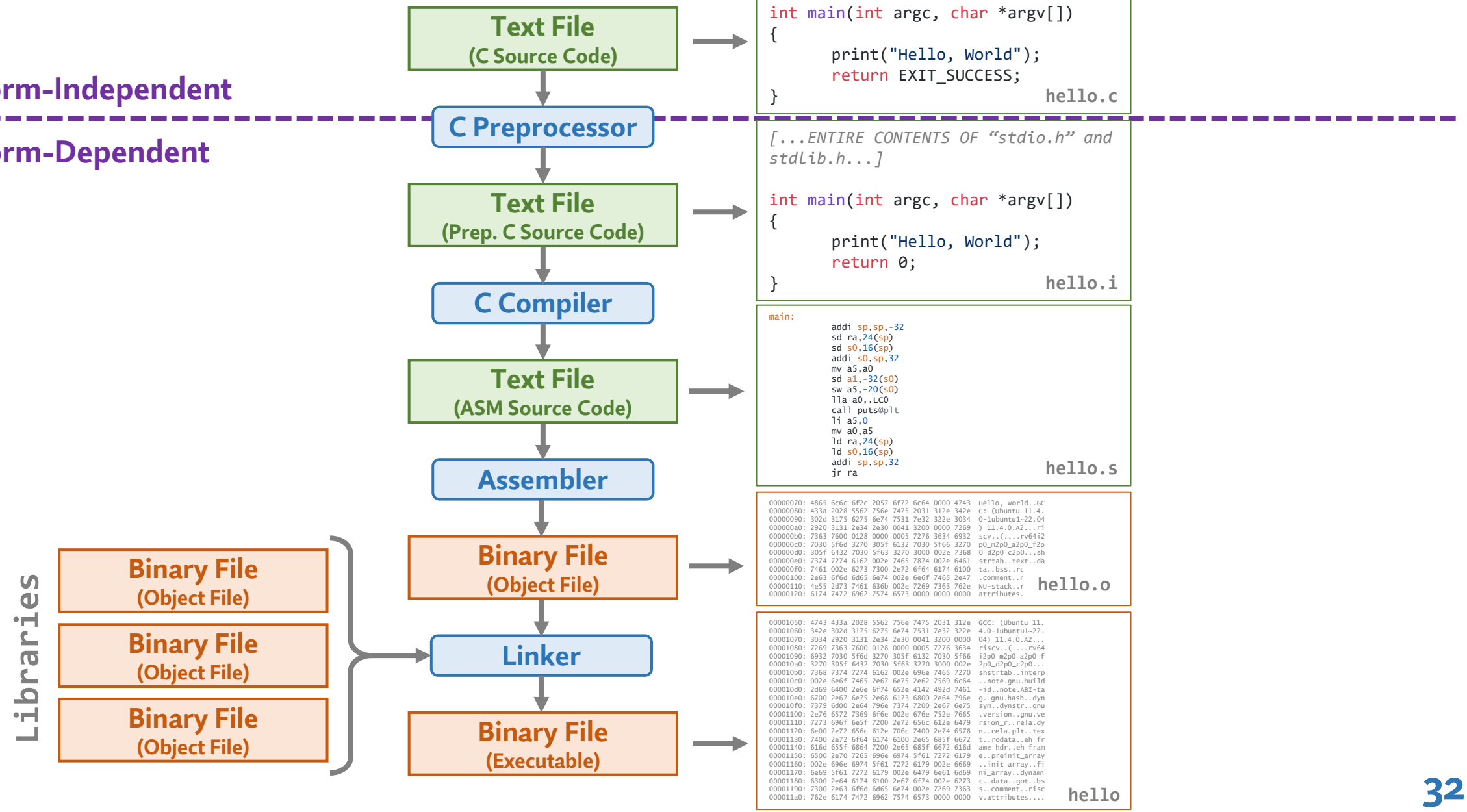


https://chatgpt.com/

Agenda

- Platforms and Compilers
- “Hello World” Analysis
- **Compiling C Code**

C Compilation

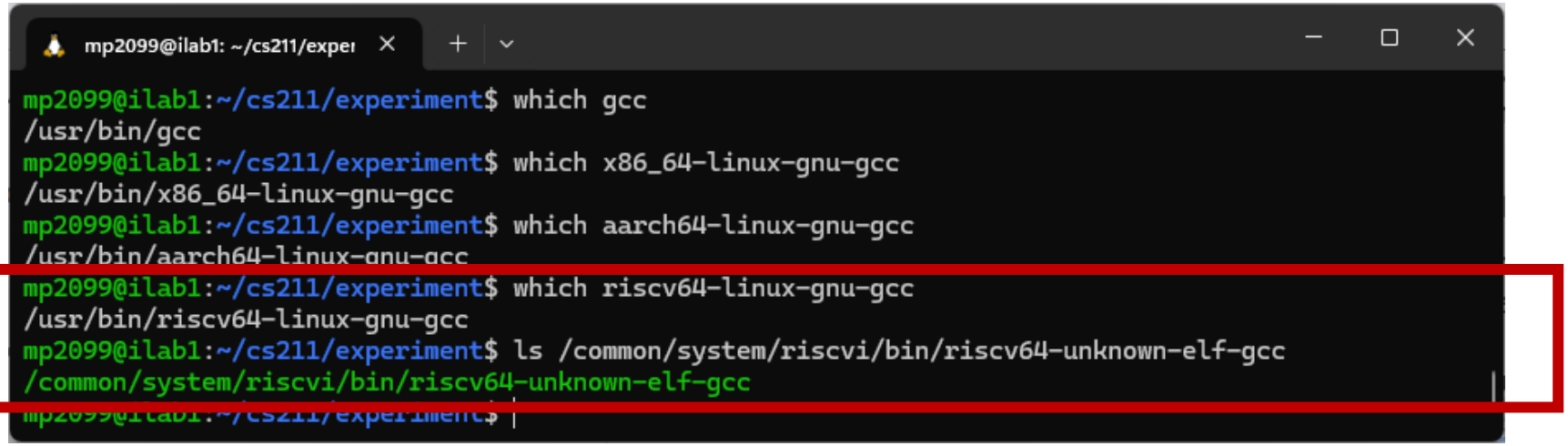


GCC: Our C Compiler Toolchain

- **GNU Compiler Collection (GCC):** A set of compilers for many different languages and platforms
  - It's (mostly) written in C and *itself needs to be compiled* to actually run

GCC variants on ilab (for different targets):

This Class



general usage:

```
netid@ilab2:~$ {optional prefix}gcc [flags] [filenames]
```

Useful GCC Flags to Try Out

Specify the output filename (default is a.out)

```
netid@ilab2:~$ riscv64-linux-gnu-gcc -o [output filename] [filenames]
```

Code optimization level

```
netid@ilab2:~$ riscv64-linux-gnu-gcc -On [filenames]
```

-O0: no optimizations on your code
-O1, -O2, -O3: optimize for performance
-Os: optimize for small machine code size
-Og: don't interfere with debugging

Warnings

```
netid@ilab2:~$ riscv64-linux-gnu-gcc -Wall [filenames]
```

-Wall: enable more warnings
-Wextra: enable even more more warnings
-Wpedantic: enable even more more warnings

Errors

```
netid@ilab2:~$ riscv64-linux-gnu-gcc -Werror [filenames]
```

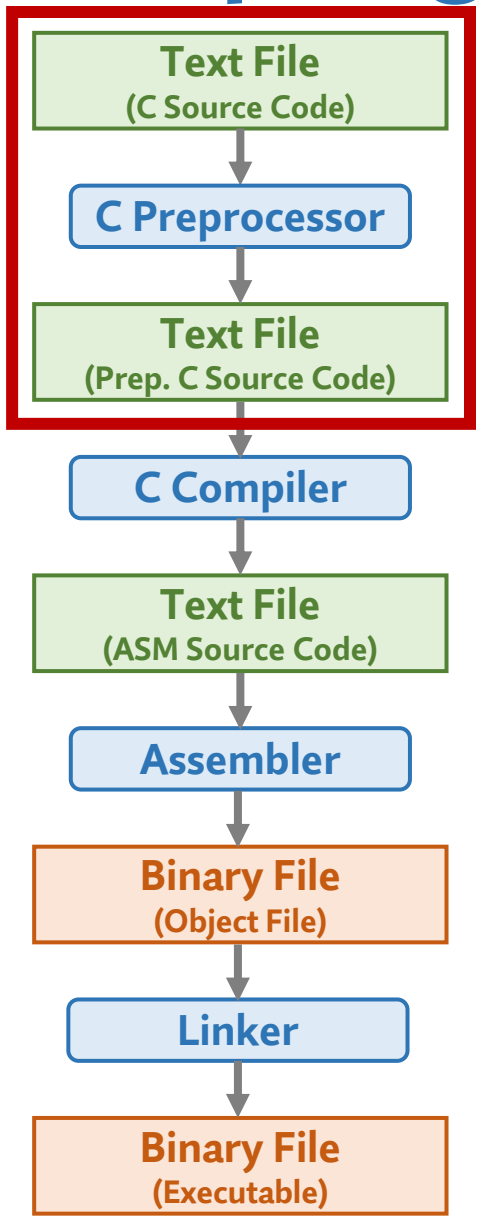
-Werror: treat warnings as errors
-Wfatal-errors: stop after the first error

Enable Debugging Metadata (e.g., for GDB)

```
netid@ilab2:~$ riscv64-linux-gnu-gcc -g [filenames]
```

https://gcc.gnu.org/onlinedocs/gcc/Option-Summary.html

Compiling the C Starter Code: Preprocessing



hello_world.c

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    // some comment here
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

C Source Code

e.g., ASCII, UTF-8

hello_world.i

```c
[Entire contents of stdlib.h: 355 lines]
[Entire contents of stdio.h: 308 lines]

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return 0;
}
```

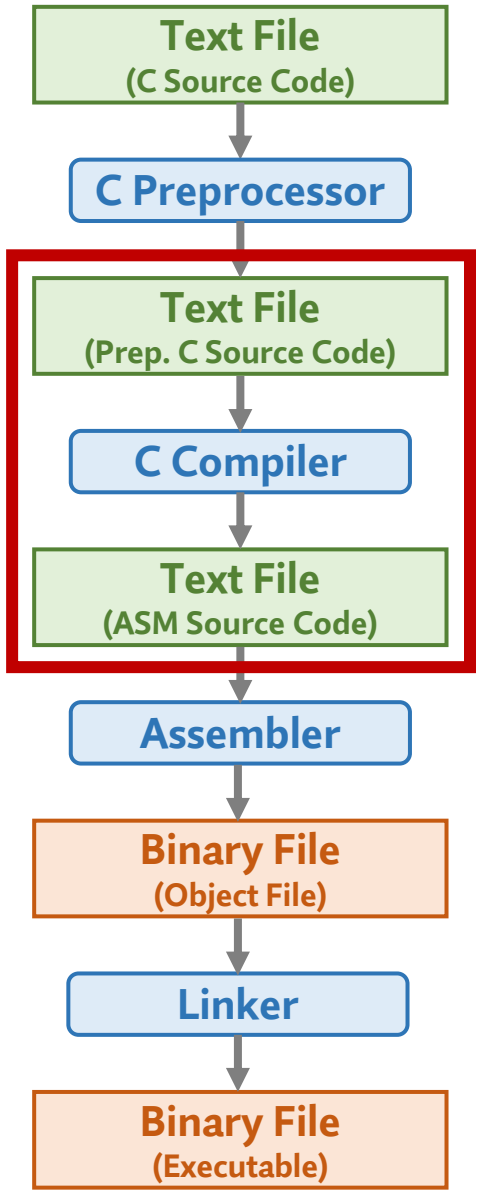
C Source Code (Preprocessed)

e.g., ASCII, UTF-8

Preprocessor: `gcc -E [flags] [filenames]`

```
mp2099@ilab4: ~/cs211/expe
mp2099@ilab4:~/cs211/experiment$ /common/system/riscvi/bin/riscv64-unknown-elf-gcc -E -P hello_world.c > hello_world.i
mp2099@ilab4:~/cs211/experiment$
```

Compiling the C Starter Code: Compilation



hello_world.i

```
[Entire contents of stdlib.h: 355 lines]
[Entire contents of stdio.h: 308 lines]

int main(int argc, char *argv[])
{
    puts("Hello World!");
    return 0;
}
```

hello_world.s

```
.LC0:
        .string "Hello World!"
main:
        addi    sp,sp,-32
        sd      ra,24(sp)
        sd      s0,16(sp)
        addi    s0,sp,32
        mv      a5,a0
        sd      a1,-32(s0)
        sw      a5,-20(s0)
        lla     a0,.LC0
        call    puts
        li      a5,0
        mv      a0,a5
        ld      ra,24(sp)
        ld      s0,16(sp)
        addi    sp,sp,32
        jr      ra
```

C Source Code (Preprocessed)

e.g., ASCII, UTF-8

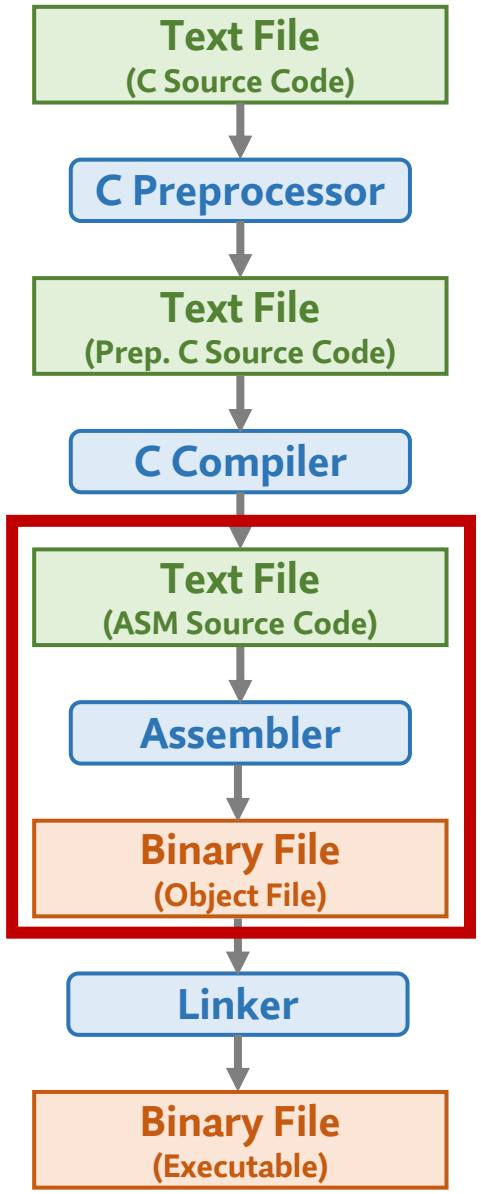
Assembler Source Code

e.g., ASCII, UTF-8

Compilation: `gcc -S [flags] [filenames]`

```
mp2099@ilab4: ~/cs211/expe
mp2099@ilab4:~/cs211/experiment$ /common/system/riscvi/bin/riscv64-unknown-elf-gcc -S -o hello_world.s hello_world.i
mp2099@ilab4:~/cs211/experiment$
```

Compiling the C Starter Code: Assembly



hello_world.s

```
        .string "Hello World!"
main:
        addi    sp,sp,-32
        sd      ra,24(sp)
        sd      s0,16(sp)
        addi    s0,sp,32
        mv      a5,a0
        sd      a1,-32(s0)
        sw      a5,-20(s0)
        lla     a0,.LC0
        call    puts
        li      a5,0
        mv      a0,a5
        ld      ra,24(sp)
        ld      s0,16(sp)
        addi    sp,sp,32
        jr      ra
```

Assembler Source Code

e.g., ASCII, UTF-8

1:1 mapping

hello_world.o

```
mp2099@ilab4:~/cs211/experiment$ file hello_world.o
hello_world.o: ELF 64-bit LSB relocatable, UCB RISC-V, RVC, double-
float ABI, version 1 (SYSV), not stripped
mp2099@ilab4:~/cs211/experiment$ xxd hello_world.o
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0100 f300 0100 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 a003 0000 0000 0000  ................
00000030: 0500 0000 4000 0000 0000 4000 0c00 0b00  ....@.....@.....
00000040: 0111 06ec 22e8 0010 aa87 2330 b4fe 2326  ....".....#0..#&
00000050: f4fe b707 0000 1385 0700 9700 0000 e780  ................
00000060: 0000 8147 3e85 e260 4264 0561 8280 0000  ...G>..`Bd.a....
00000070: 4865 6c6c 6f20 576f 726c 6421 0000 4743  Hello World!..GC
00000080: 433a 2028 2920 3134 2e32 2e30 0041 6500  C: () 14.2.0.Ae.
00000090: 0000 7269 7363 7600 015b 0000 0004 1005  ..riscv..[......
000000a0: 7276 3634 6932 7031 5f6d 3270 305f 6132  rv64i2p1_m2p0_a2
000000b0: 7031 5f66 3270 325f 6432 7032 5f63 3270  p1_f2p2_d2p2_c2p
000000c0: 305f 7a69 6373 7232 7030 5f7a 6966 656e  0_zicsr2p0_zifen
```

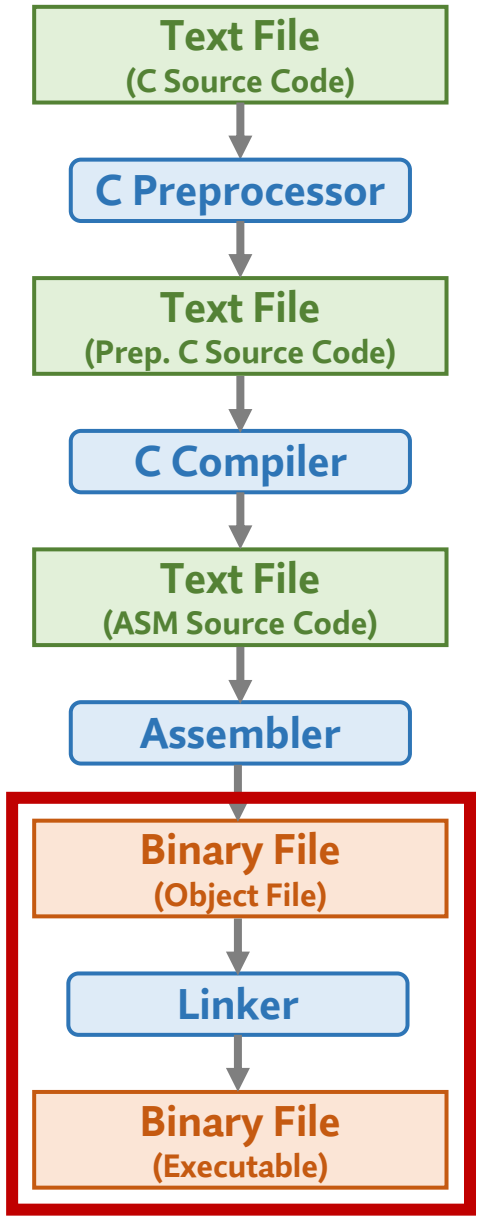
Object File

ELF Format (Binary)

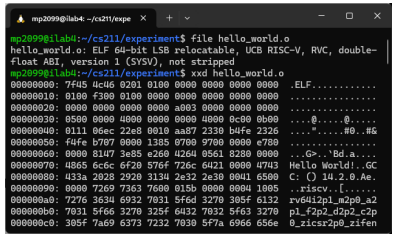
Assembly: `gcc -c [flags] [filenames]`

```
mp2099@ilab4:~/cs211/experiment$ /common/system/riscvi/bin/riscv64-unknown-elf-gcc -c -o hello_world.o hello_world.s
mp2099@ilab4:~/cs211/experiment$
```

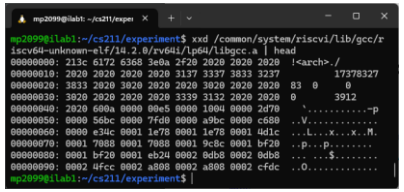
Compiling the C Starter Code: Linking



hello_world.o



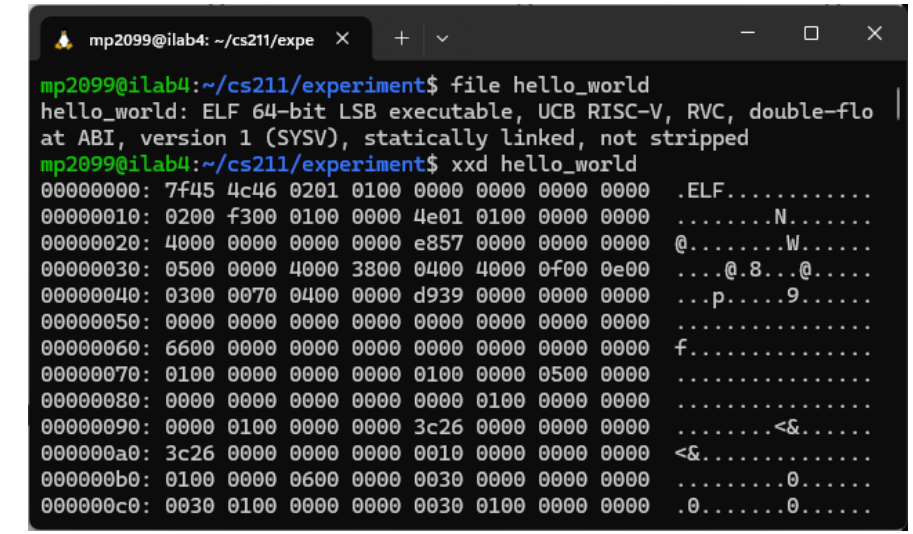
C Standard Library



Object Files

ELF Format (Binary)

hello_world

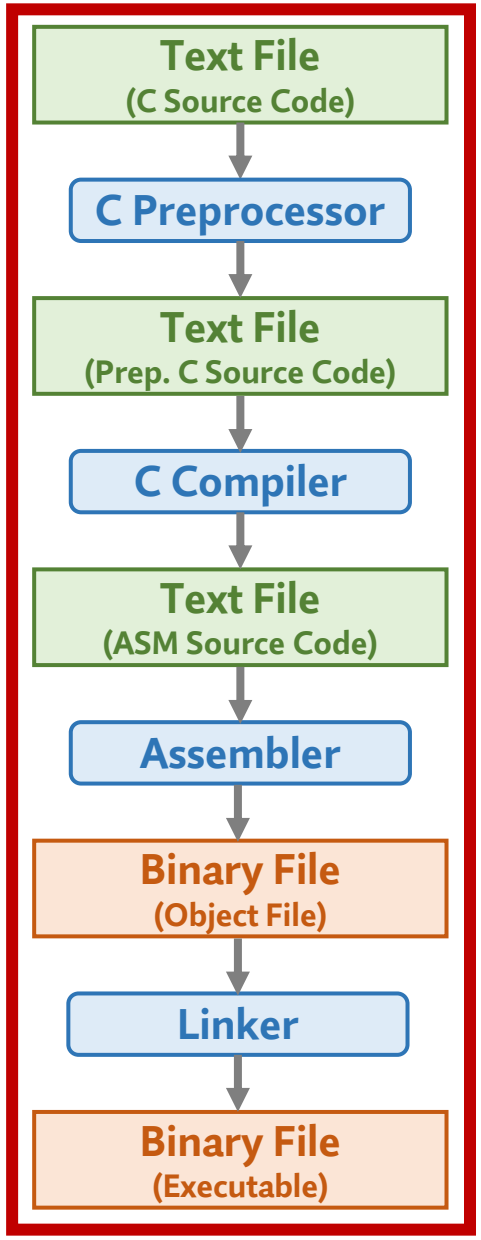


Executable Binary

ELF Format (Binary)

Linker: `gcc -o [exename] [flags] [filenames]`

Compiling the C Starter Code: All in One Step



hello_world.c

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

C Source Code

e.g., ASCII, UTF-8

hello_world

```
mp2099@ilab4:~/cs211/experiment$ file hello_world
hello_world: ELF 64-bit LSB executable, UCB RISC-V, RVC, double-flo
at ABI, version 1 (SYSV), statically linked, not stripped
mp2099@ilab4:~/cs211/experiment$ xxd hello_world
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0200 f300 0100 0000 4e01 0100 0000 0000  ........N.......
00000020: 4000 0000 0000 0000 e857 0000 0000 0000  @........W......
00000030: 0500 0000 4000 3800 0400 4000 0f00 0e00  ....@.8...@.....
00000040: 0300 0070 0400 0000 d939 0000 0000 0000  ...p.....9......
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 6600 0000 0000 0000 0000 0000 0000 0000  f...............
00000070: 0100 0000 0000 0000 0100 0000 0500 0000  ................
00000080: 0000 0000 0000 0000 0000 0100 0000 0000  ................
00000090: 0000 0100 0000 0000 3c26 0000 0000 0000  ........<&......
000000a0: 3c26 0000 0000 0000 0010 0000 0000 0000  <&..............
000000b0: 0100 0000 0600 0000 0030 0000 0000 0000  .........0......
000000c0: 0030 0100 0000 0000 0030 0100 0000 0000  .0.......0......
```

Executable Binary

ELF Format (Binary)

One-shot: `gcc -o [exename] [flags] [C filenames]`

```
mp2099@ilab4:~/cs211/experiment$ /common/system/riscvi/bin/riscv64-unknown-elf-gcc -o hello_world hello_world.c
mp2099@ilab4:~/cs211/experiment$ ./hello_world
Hello World!
mp2099@ilab4:~/cs211/experiment$
```

GCC Demo

- (Time Permitting)

CS 211: Intro to Computer Architecture

4.2: C Preprocessing and Compilation

Minesh Patel

Spring 2025 – Thursday 13 February