

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

The screenshot shows the website **cppreference.com**. At the top, there is a search bar and a "Create account" link. Below the search bar, there are tabs for "Page" and "Discussion", and a "Standard revision: Diff" dropdown menu. The main content area is titled "C reference" and lists various C-related topics under the heading "C". The topics are organized into three columns:

- Language**: Basic concepts, Keywords, Preprocessor, Expressions, Declaration, Initialization, Functions, Statements.
- Program utilities**: Variadic functions, Diagnostics library, Dynamic memory management, Strings library (Null-terminated strings: byte – multibyte – wide), Date and time library, Localization library, Input/output library.
- Algorithms library**: Numerics library (Common mathematical functions, Floating-point environment (C99), Pseudo-random number generation, Complex number arithmetic (C99), Type-generic math (C99), Bit manipulation (C23), Checked integer arithmetic (C23)), Concurrency support library (C11).

The screenshot shows the cover page of the International Standard **ISO/IEC 9899:201x**. At the top, it says "N1570 Committee Draft — April 12, 2011". Below this, it says "INTERNATIONAL STANDARD ©ISO/IEC ISO/IEC 9899:201x". The main title is "Programming languages — C". At the bottom, the word "ABSTRACT" is visible.

```
mp2099@ilab4: ~/cs211/expe
MAN(1) Manual pager utils MAN(1)
NAME
man - an interface to the system reference manuals
Manual page man(1) line 1 (press h for help or q to quit)
```

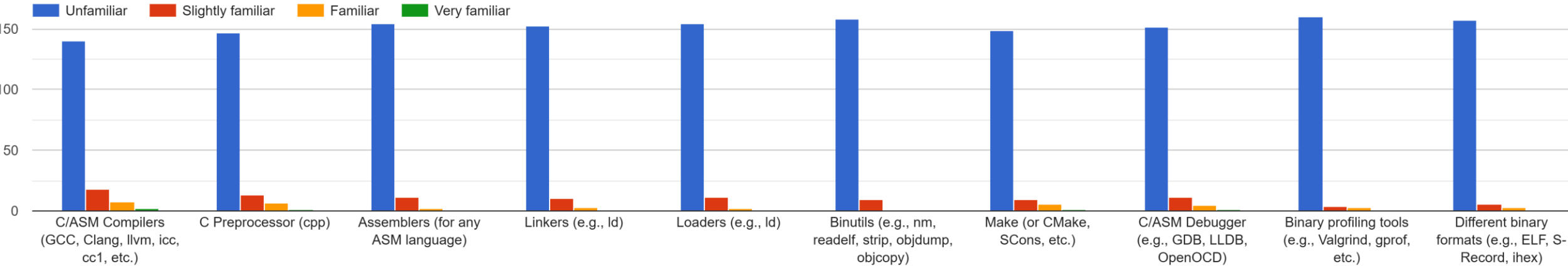
Shout-out to Related Material

- “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-AC_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
    armb, // ARM (big endian): armb
    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)
    bpfbe, // 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, mipsallegregex, mipsr6
    mipsel, // MIPS EL: mipsel, mipsallegrexe, mipsr6el
    mips64, // MIPS64: mips64, mips64r6, mipsn32, mipsn32r6
    mips64el, // MIPS64 EL: mips64el, mips64r6el, mipsn32el, mipsn32r6el
    msp430, // MSP430: msp430
    ppc, // PPC: powerpc
    ppcle, // PPC LE: powerpc (little endian)
    ppc64, // PPC64: powerpc64, ppu
    ppc64le, // PPC64 LE: powerpc64le
    r600, // R600: AMD GPUs HD2XXX - HD6XXX
    amdgc, // AMDGCN: AMD GCN GPUs
    riscv32, // RISC-V (32-bit): riscv32
    riscv64, // RISC-V (64-bit): riscv64
    sparc, // Sparc: sparc
    sparcv9, // Sparc v9: 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.*
    thumbbe, // Thumb (big endian): thumbbe
    x86, // X86: i[3-8]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,
    ELFAMCU,
    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,
    SCÉI,
    Freescale,
    IBM,
    ImaginationTechnologies,
    MipsTechnologies,
    NVIDIA,
    CSR,
    AMD,
    Mesa,
    SUSE,
    OpenEmbedded,
    Intel,
    LastVendorType = Intel
};
```

```
enum EnvironmentType {
    UnknownEnvironment,

    GNU,
    GNU64,
    GNUABI32,
    GNUABI64,
    GNUEABI,
    GNUEABI64,
    GNUEABIHF,
    GNUEABIHF64,
    GNUF32,
    GNUF64,
    GNUSF,
    GNUX32,
    GNUILP32,
    CODE16,
    EABI,
    EABIHF,
    Android,
    Musl,
    MuslABI32,
    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
};
```


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

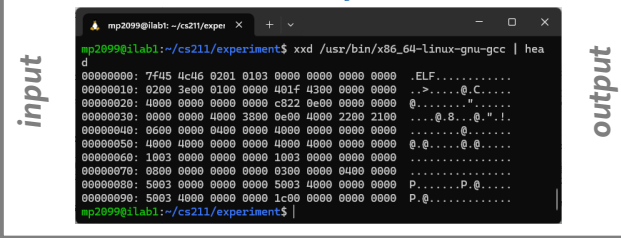
C Source Code (Platform-Independent)

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

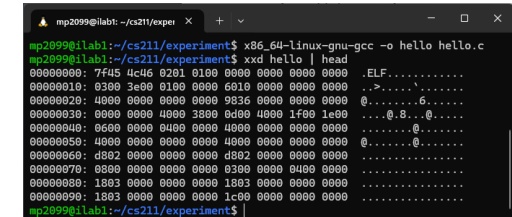
int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

Machine Readable

Executable Binary (C Compiler) (Platform-Dependent)



Machine Readable Executable Binary (Your App) (Platform-Dependent)

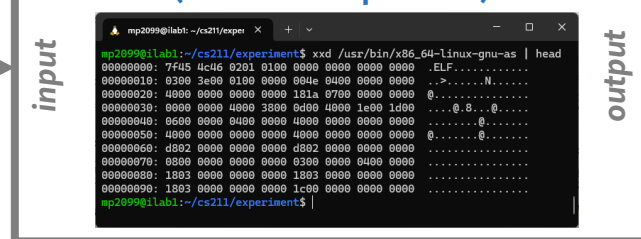


Sort-of Human Readable Assembly Source Code (Platform-Dependent)

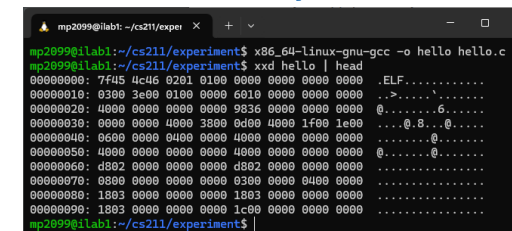
```
main:    .string "Hello World!"
        addi    sp,sp,-32
        sd     ra,24(sp)
        sd     s0,16(sp)
        addi    s0,sp,32
        mv     a5,a0
        sd     a1,-32(s0)
        sw     a5,-20(s0)
        lla    a0,.LC0
        call   puts
        li     a5,0
        mv     a0,a5
      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

Machine Readable

Executable Binary (Assembler) (Platform-Dependent)



Machine Readable Executable Binary (Your App) (Platform-Dependent)



Compilers vs... Other Ways to Run Code

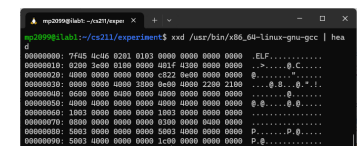
C Source Code (Platform-Independent)

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

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

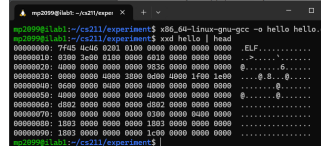
Executable Binary (C Compiler) (Platform-Dependent)

input



output

Executable Binary (Your App) (Platform-Dependent)

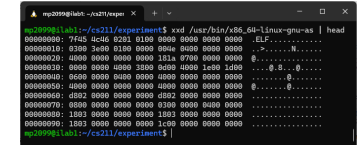


Assembly Source Code (Platform-Dependent)

```
main: .string "Hello World!"
      addi sp,sp,-32
      rd 24(sp)
      addi sp,sp,32
      mv a1,-32($0)
      sw a1,-32($0)
      lla a6,.LC0
      li a5,0
      mv rd,rd
      rd 24(sp)
      addi sp,sp,32
      jr ra
```

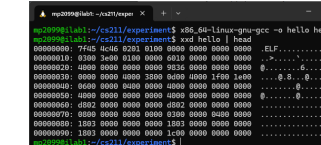
Executable Binary (Assembler) (Platform-Dependent)

input



output

Executable Binary (Your App) (Platform-Dependent)

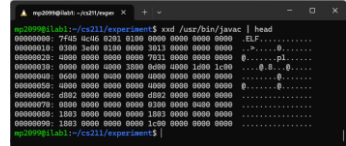


Java Source Code (Platform-Independent)

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

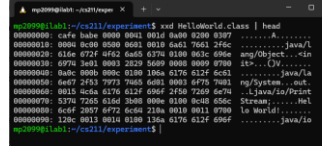
Executable Binary (javac) (Platform-Dependent)

input



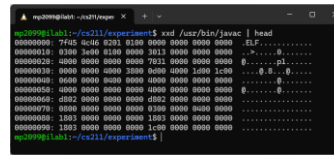
output

Java Bytecode (Platform-Independent)



Executable Binary (JVM) (Platform-Dependent)

input



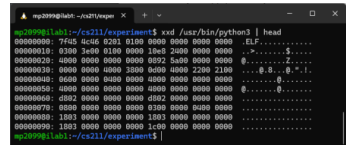
Python Source Code (Platform-Independent)

```
import os

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

Executable Binary (Python Interpreter) (Platform-Dependent)

input



“Libraries” are Executable Code

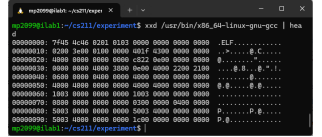
Numpy Source Code (Platform-Independent)

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

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

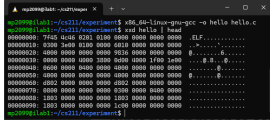
Executable Binary (C Compiler) (Platform-Dependent)

input



output

Executable Binary (NumPy Library) (Platform-Dependent)

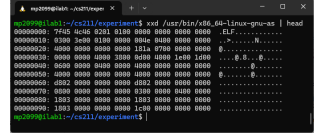


NumPy Source Code (Platform-Dependent)

```
main:
.string "Hello World!"
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 s0,lC0
call puts
li a5,0
mv a0,a5
ld ra,24(sp)
ld s0,16(sp)
addi sp,sp,32
jr ra
```

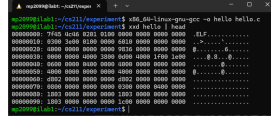
Executable Binary (Assembler) (Platform-Dependent)

input



output

Executable Binary (NumPy Library) (Platform-Dependent)



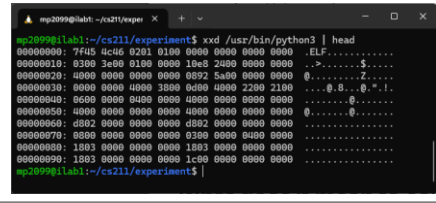
Python Source Code (Platform-Independent)

```
import numpy as np

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

Executable Binary (Python Interpreter) (Platform-Dependent)

input



output

Compiler "Targets"

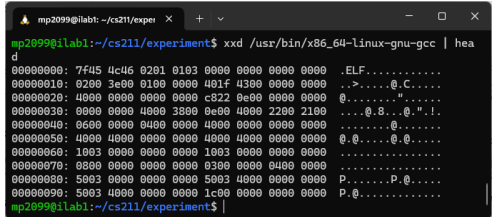
Source Code
(Platform-Independent)

```
#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)



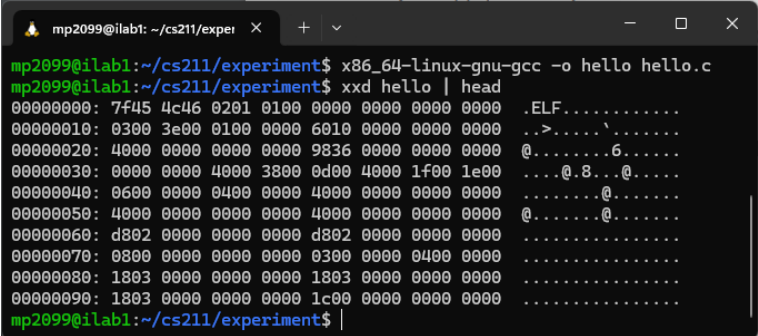
input *output*

"native" compiler target:
same platform as the host

"cross" compiler target:
different platform from the host

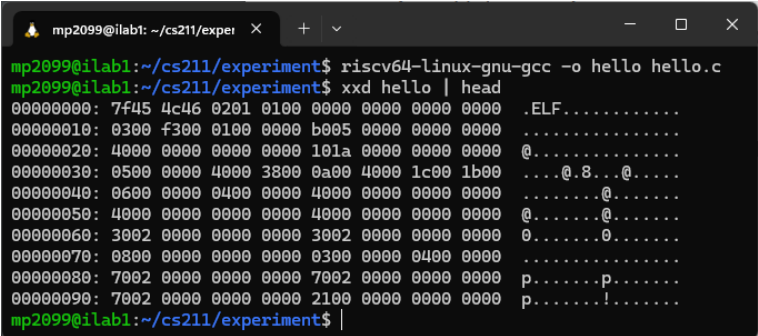
Executable Binaries
(Platform-Dependent)

Running on x86_64-linux



```
mp2099@ilab1: ~/cs211/exper x + -
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  ..>.....@.C.....
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$ |
```

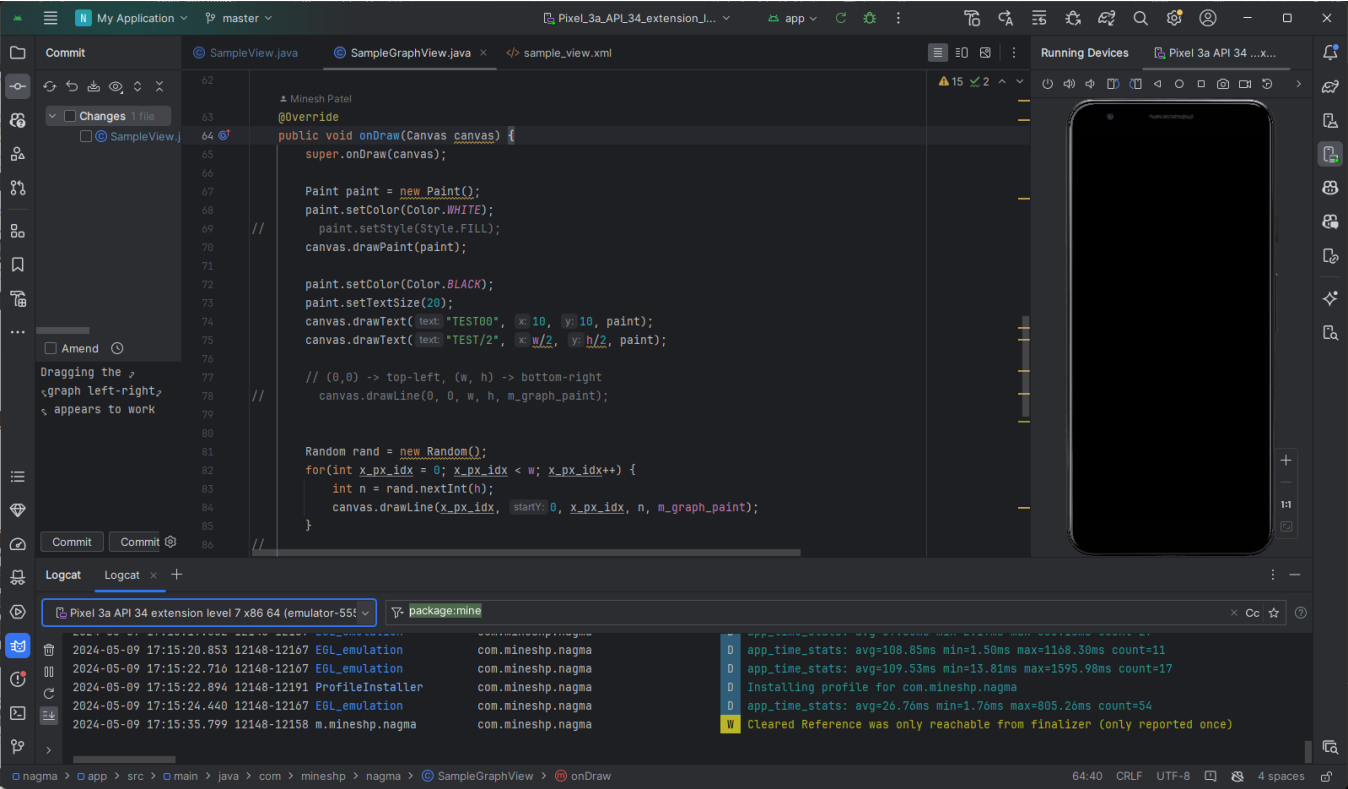
Running on riscv64-linux



```
mp2099@ilab1: ~/cs211/exper x + -
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  ..>.....@.C.....
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  .....@.....
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

```
mp2099@ilab4: ~/cs211/expe x + v - □ x
mp2099@ilab4:~/cs211/experiment$ x86_64-linux-gnu-gcc -c -o hello.o hello.c
mp2099@ilab4:~/cs211/experiment$ file hello.o
hello.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped
mp2099@ilab4:~/cs211/experiment$ |
```

Targeting
X86_64-linux

This Class

```
mp2099@ilab4: ~/cs211/expe x + v - □ x
mp2099@ilab4:~/cs211/experiment$ riscv64-linux-gnu-gcc -c -o hello.o hello.c
mp2099@ilab4:~/cs211/experiment$ file hello.o
hello.o: ELF 64-bit LSB relocatable, UCB RISC-V, RVC, double-float ABI, versi
on 1 (SYSV), not stripped
mp2099@ilab4:~/cs211/experiment$ |
```

Targeting
riscv64-linux

```
mp2099@ilab4: ~/cs211/expe x + v - □ x
mp2099@ilab4:~/cs211/experiment$ aarch64-linux-gnu-gcc -c -o hello.o hello.c
mp2099@ilab4:~/cs211/experiment$ file hello.o
hello.o: ELF 64-bit LSB relocatable, ARM aarch64, version 1 (SYSV), not strip
ped
mp2099@ilab4:~/cs211/experiment$ |
```

Targeting
aarch64-linux

Agenda

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

C vs. Java “Hello World”

hello_world.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

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


Preprocessor Directives

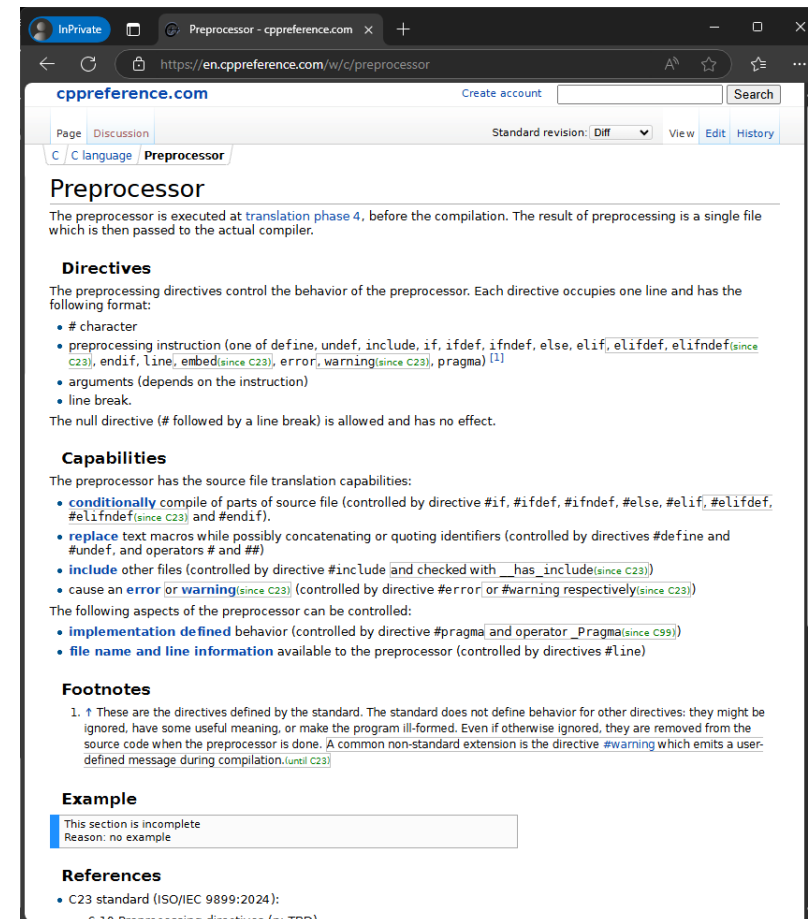
“**#include**” is a **preprocessor directive**

- 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**

hello_world.c

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

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



cppreference.com

Preprocessor

The preprocessor is executed at [translation phase 4](#), before the compilation. The result of preprocessing is a single file which is then passed to the actual compiler.

Directives

The preprocessing directives control the behavior of the preprocessor. Each directive occupies one line and has the following format:

- # character
- preprocessing instruction (one of [define](#), [undef](#), [include](#), [if](#), [ifdef](#), [ifndef](#), [else](#), [elif](#), [elifdef](#), [elifndef](#) (since C23), [endif](#), [line](#), [embed](#) (since C23), [error](#), [warning](#) (since C23), [pragma](#)) ^[1]
- arguments (depends on the instruction)
- line break.

The null directive (# followed by a line break) is allowed and has no effect.

Capabilities

The preprocessor has the source file translation capabilities:

- **conditionally** compile of parts of source file (controlled by directive [#if](#), [#ifdef](#), [#ifndef](#), [#else](#), [#elif](#), [#elifdef](#), [#elifndef](#) (since C23) and [#endif](#)).
- **replace** text macros while possibly concatenating or quoting identifiers (controlled by directives [#define](#) and [#undef](#), and operators <#> and [##](#))
- **include** other files (controlled by directive [#include](#) and checked with [_has_include](#) (since C23))
- cause an **error** or **warning** (since C23) (controlled by directive [#error](#) or [#warning](#) respectively (since C23))

The following aspects of the preprocessor can be controlled:

- **implementation defined** behavior (controlled by directive [#pragma](#) and operator [_Pragma](#) (since C99))
- **file name and line information** available to the preprocessor (controlled by directives [#line](#))

Footnotes

1. ↑ These are the directives defined by the standard. The standard does not define behavior for other directives: they might be ignored, have some useful meaning, or make the program ill-formed. Even if otherwise ignored, they are removed from the source code when the preprocessor is done. A common non-standard extension is the directive [#warning](#) which emits a user-defined message during compilation. (since C23)

Example

This section is incomplete
Reason: no example

References

- C23 standard (ISO/IEC 9899:2024):
 - 6.10 Preprocessing directives (e.g. TR1)

Preprocessor Directives

stdlib.h Source Code (yes, they use Git too!)

hello_world.c

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

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



```
1 /* Copyright (C) 1991-2025 Free Software Foundation, Inc.
2 Copyright The GNU Toolchain Authors.
3 This file is part of the GNU C Library.
4
5 The GNU C Library is free software; you can redistribute it and/or
6 modify it under the terms of the GNU Lesser General Public
7 License as published by the Free Software Foundation; either
8 version 2.1 of the License, or (at your option) any later version.
9
10 The GNU C Library is distributed in the hope that it will be useful,
11 but WITHOUT ANY WARRANTY; without even the implied warranty of
12 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
13 Lesser General Public License for more details.
14
15 You should have received a copy of the GNU Lesser General Public
16 License along with the GNU C Library; if not, see
17 <https://www.gnu.org/licenses/>. */
18
19 /*
20 * ISO C99 Standard: 7.20 General utilities <stdlib.h>
21 */
22
23 #ifndef _STDLIB_H
24
25 #define __GLIBC_INTERNAL_STARTING_HEADER_IMPLEMENTATION
26 #include <bits/libc-header-start.h>
27
28 /* Get size_t, wchar_t and NULL from <stddef.h>. */
29 #define __need_size_t
30 #define __need_wchar_t
31 #define __need_NULL
32 #include <stddef.h>
```

“#define” is another
preprocessor directive

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

```
90 /* We define these the same for all machines.
91 Changes from this to the outside world should be done in `__exit'. */
92 #define EXIT_FAILURE 1 /* Failing exit status. */
93 #define EXIT_SUCCESS 0 /* Successful exit status. */
94
```

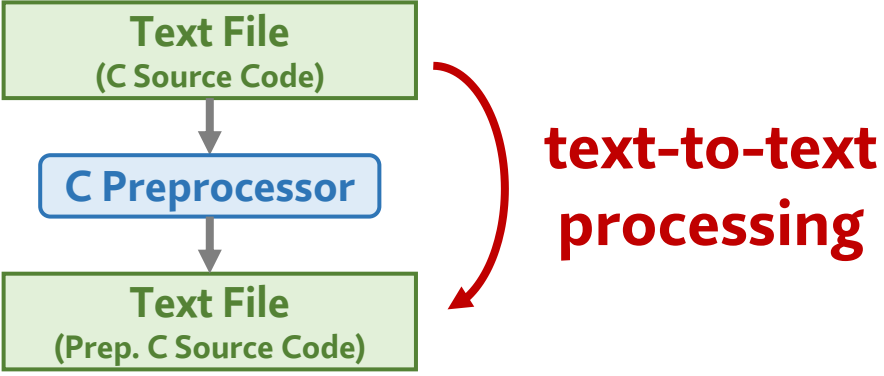
preprocessor macro

C Preprocessing

hello_world.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

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

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

C Standard Library: stdlib.h Documentation

Reference : <stdlib>

Ad served by Google

header

<stdlib> (stdlib.h)

C Standard General Utilities Library

This header defines several general purpose functions, including dynamic memory management, random number generation, communication with the environment, integer arithmetics, searching, sorting and converting.

Functions

String conversion

atof	Convert string to double (function)
atoi	Convert string to integer (function)
atol	Convert string to long integer (function)
atoll	Convert string to long long integer (function)
strtod	Convert string to double (function)
strtof	Convert string to float (function)
strtol	Convert string to long integer (function)
strtold	Convert string to long double (function)
strtoll	Convert string to long long integer (function)
strtoul	Convert string to unsigned long integer (function)
strtoull	Convert string to unsigned long long integer (function)

Pseudo-random sequence generation

rand	Generate random number (function)
srand	Initialize random number generator (function)

Dynamic memory management

calloc	Allocate and zero-initialize array (function)
------------------------	---

Ad served by Google

<stdio.h>

hello_world.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

The screenshot shows a web browser window displaying the documentation for the C standard library header `<stdio.h>` on the Cplusplus.com website. The page title is "Reference : <stdio.h>". The left sidebar contains a navigation menu with "C++", "Tutorials", "Reference", "Articles", and "Forum". The main content area is titled "C library to perform Input/Output operations" and includes an introduction to the C Standard Input and Output Library (`stdio.h`), a section on "Stream properties", and a section on "Read/Write Access". The page also features a Google advertisement and a search bar.

Reference : <stdio.h>

Ads by Google

Send feedback Why this ad? ▾

header

<stdio.h> (stdio.h)

C library to perform Input/Output operations

Input and Output operations can also be performed in C++ using the C Standard Input and Output Library (`stdio.h`, known as `stdio.h` in the C language). This library uses what are called **streams** to operate with physical devices such as keyboards, printers, terminals or with any other type of files supported by the system. Streams are an abstraction to interact with these in a uniform way. All streams have similar properties independently of the individual characteristics of the physical media they are associated with.

Streams are handled in the `stdio` library as pointers to [FILE](#) objects. A pointer to a [FILE](#) object uniquely identifies a stream, and is used as a parameter in the operations involving that stream.

There also exist three standard streams: [stdin](#), [stdout](#) and [stderr](#), which are automatically created and opened for all programs using the library.

Stream properties

Streams have some properties that define which functions can be used on them and how these will treat the data input or output through them. Most of these properties are defined at the moment the stream is associated with a file (opened) using the [fopen](#) function:

Read/Write Access

Specifies whether the stream has read or write access (or both) to the physical media they are associated with.

Text / Binary

Text streams are thought to represent a set of text lines, each one ending with a new-line character. Depending on the environment where the application is run, some character translation may occur with text streams to adapt some special characters to the text file specifications of the environment. A binary stream, on the other hand, is a sequence of characters written or read from the physical media with no translation, having a one-to-one correspondence with the characters read or written to the stream.

Buffer

A buffer is a block of memory where data is accumulated before being physically read or written to the associated file or device. Streams can be either **fully buffered**, **line buffered** or **unbuffered**. On fully buffered streams, data is read/written when the buffer is filled, on line buffered streams...

C Standard Libraries

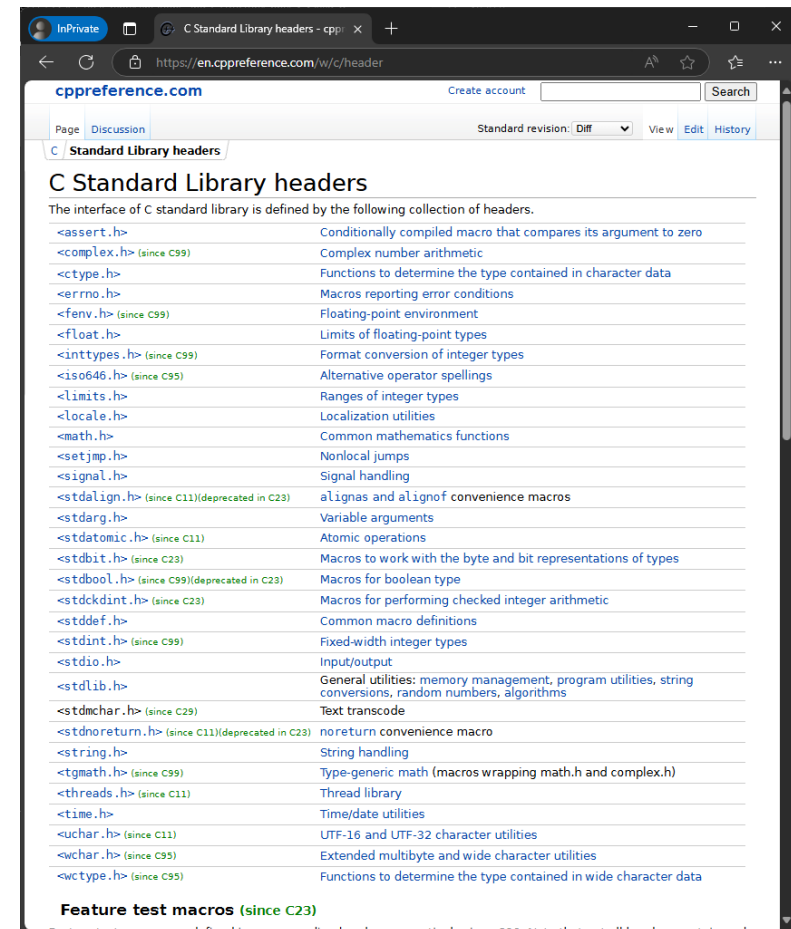
- **C Standard Libraries** are... standardized
 - Basic utility functions and macros
 - Required by the C standard

hello_world.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

```
#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

```
#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
```

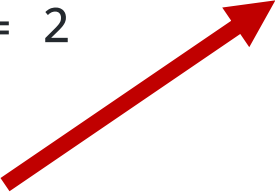
```
argc = 4
```

```
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

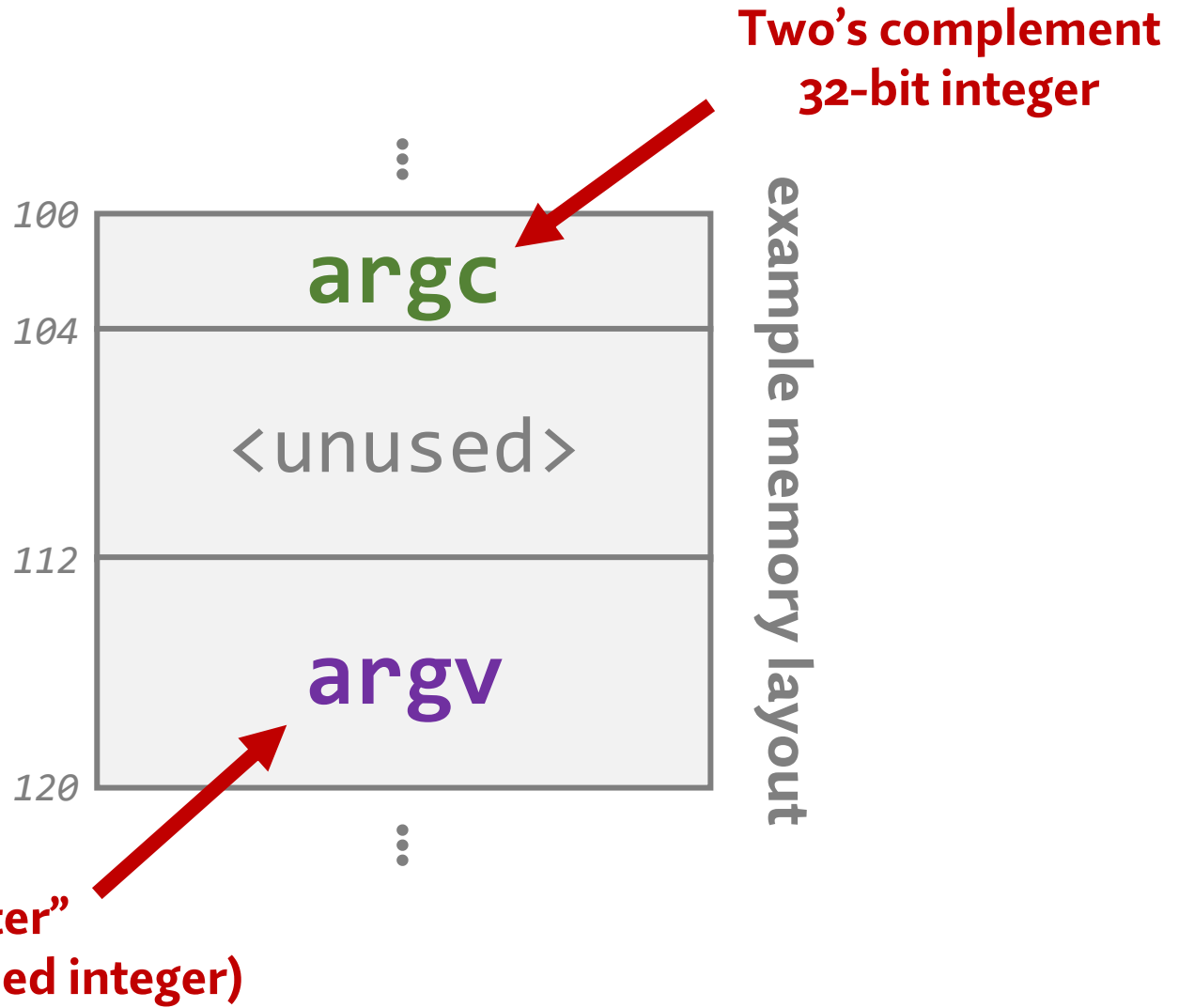
C Starter Code: Objects in Memory

hello_world.c

```
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
    puts("Hello World!");
    return EXIT_SUCCESS;
}
```

objects

int argc **char ***argv[]

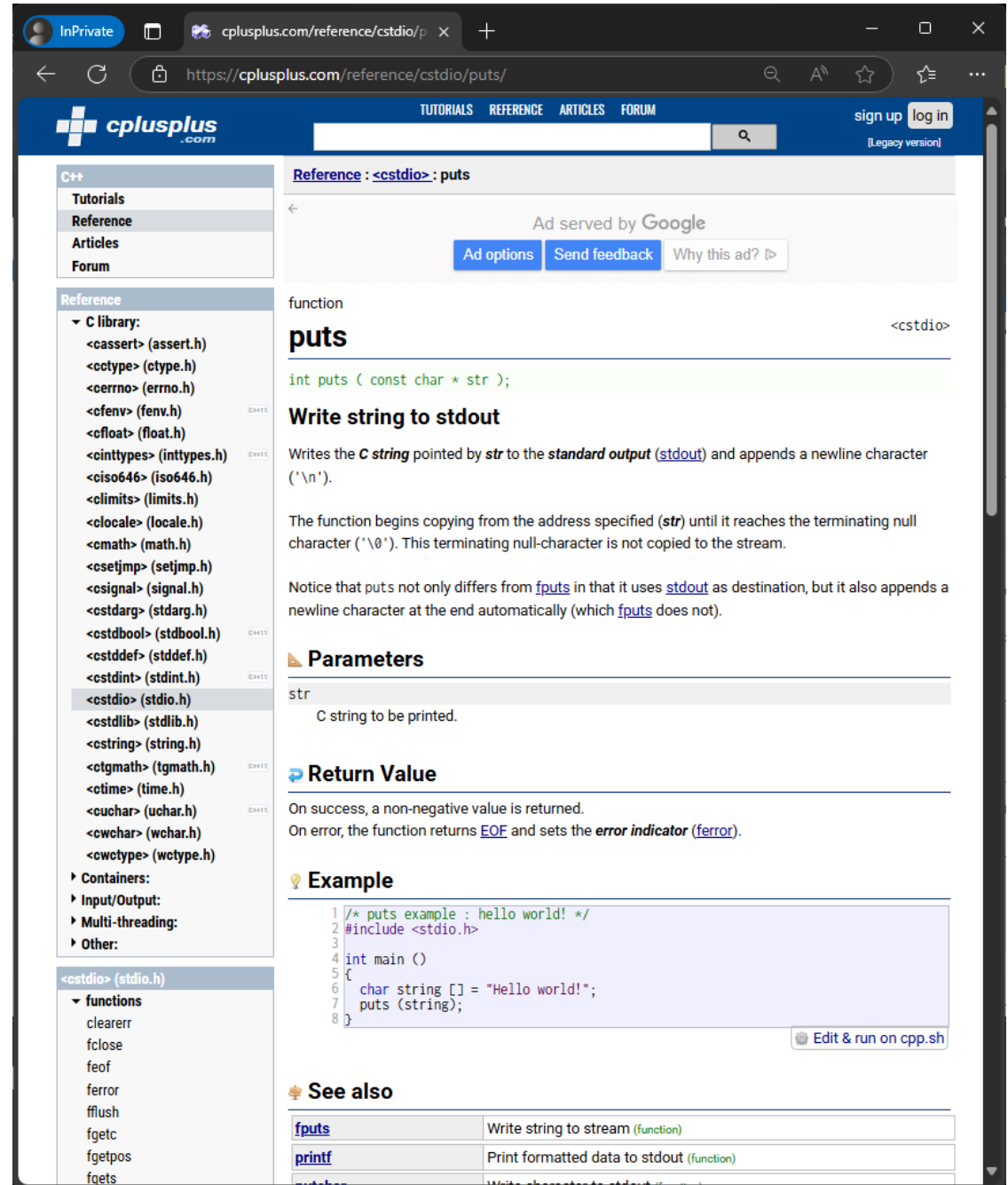


C Starter Code: puts

hello_world.c

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

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



The screenshot shows a web browser window displaying the reference page for the `puts` function on the cplusplus.com website. The page title is "Reference : <stdio> : puts". The left sidebar contains a navigation menu with "C++", "Tutorials", "Reference", "Articles", and "Forum". The main content area includes an advertisement, the function signature `int puts (const char * str);`, a description of the function's purpose (writing a string to stdout and appending a newline), and a list of parameters (the `str` parameter). The page also features sections for "Return Value" (a non-negative value on success, EOF on error) and "Example" (a code snippet demonstrating the use of `puts` to print "Hello world!").

C Starter Code: puts

```
mp2099@ilab1: ~/cs211, x + v - □ x
mp2099@ilab1:~/cs211/experiment$ man puts|
```



hello_world.c

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

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

```
mp2099@ilab1: ~/cs211/exper x + v - □ x
PUTS(3) Linux Programmer's Manual PUTS(3)
NAME
    fputc, fputs, putc, putchar, puts - output of characters and strings
SYNOPSIS
    #include <stdio.h>
    int fputc(int c, FILE *stream);
    int fputs(const char *s, FILE *stream);
    int putc(int c, FILE *stream);
    int putchar(int c);
    int puts(const char *s);
DESCRIPTION
    fputc() writes the character c, cast to an unsigned char, to stream.
    fputs() writes the string s to stream, without its terminating null byte ('\0').
    putc() is equivalent to fputc() except that it may be implemented as a macro which evaluates stream more than once.
    putchar(c) is equivalent to putc(c, stdout).
    puts() writes the string s and a trailing newline to stdout.
Manual page puts(3) line 1 (press h for help or q to quit)
```

C Starter Code: puts

hello_world.c

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

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

what is puts in c

In C, `puts` is a standard library function used to output a string to the console, followed by a newline character (`\n`). It is declared in the `stdio.h` header file.

Syntax:

```
int puts(const char *str);
```

Parameters:

- `str`: A pointer to the null-terminated string you want to print.

Return Value:

- On success, `puts` returns a non-negative number.
- On failure, it returns `EOF` (End Of File).

Example:

```
#include <stdio.h>

int main() {
    puts("Hello, world!");
    return 0;
}
```

Agenda

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

C Compilation

Platform-Independent

Platform-Dependent

Libraries

Text File
(C Source Code)

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

int main(int argc, char *argv[])
{
    print("Hello, World");
    return EXIT_SUCCESS;
}
hello.c
```

C Preprocessor

```
[...ENTIRE CONTENTS OF "stdio.h" and
stdlib.h...]

int main(int argc, char *argv[])
{
    print("Hello, World");
    return 0;
}
hello.i
```

Text File
(Prep. C Source Code)

C Compiler

Text File
(ASM Source Code)

```
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@plt
    li a5,0
    mv a0,a5
    ld ra,24(sp)
    ld s0,16(sp)
    addi sp,sp,32
    jr ra
hello.s
```

Assembler

Binary File
(Object File)

```
00000070: 4865 6c6c 6f2c 2057 6f72 6c64 0000 4743 Hello, World..gc
00000080: 433a 2028 5562 756e 7475 2031 312e 342e c: (Ubuntu 11.4.
00000090: 302d 3175 6275 6e74 7531 7e32 322e 3034 0-lubuntu1-22.04
000000a0: 2920 3131 2e34 2e30 0041 3200 0000 7269 .) 11.4.0.A2...ri
000000b0: 7363 7600 0128 0000 0005 7276 3634 6932 scv,...rv6412
000000c0: 7030 5f6d 3270 305f 6132 7030 5f66 3270 p0_m2p0_a2p0_f2p
000000d0: 305f 6432 7030 5f63 3270 3000 002e 7368 0_d2p0_c2p0...sh
000000e0: 7374 7274 6162 002e 7465 7874 002e 6461 strtab..text..da
000000f0: 7461 002e 6273 7300 2e72 6f64 6174 6100 ta..bss..rc
00000100: 2e63 6f6d 6d65 6e74 002e 6e6f 7465 2e47 .comment..r
00000110: 4e55 2d73 7461 636b 002e 7269 7363 762e Nu-stack...r
00000120: 6174 7472 6962 7574 6573 0000 0000 0000 attributes.
```

Binary File
(Object File)

Binary File
(Object File)

Binary File
(Object File)

Linker

Binary File
(Executable)

```
00001050: 4743 433a 2028 5562 756e 7475 2031 312e gcc: (Ubuntu 11.
00001060: 342e 302d 3175 6275 6e74 7531 7e32 322e 4.0-lubuntu1-22.
00001070: 3034 2920 3131 2e34 2e30 0041 3200 0000 04) 11.4.0.A2...
00001080: 7269 7363 7600 0128 0000 0005 7276 3634 riscv,...rv64
00001090: 6932 7030 5f6d 3270 305f 6132 7030 5f66 12p0_m2p0_a2p0_f
000010a0: 3270 305f 6432 7030 5f63 3270 3000 002e 2p0_d2p0_c2p0...
000010b0: 7368 7374 7274 6162 002e 696e 7465 7270 shstrtab..interp
000010c0: 002e 6e6f 7465 2e67 6e75 2e62 7369 6c64 ..note.gnu.build
000010d0: 2d69 6400 2e6e 6f74 652e 4142 492d 7461 -fd..note.ABI-ta
000010e0: 6700 2e67 6e75 2e68 6173 6800 2e64 796e g..gnu.hash..dyn
000010f0: 7379 6d00 2e64 796e 7374 7200 2e67 6e75 sym..dynstr..gnu
00001100: 2e76 6572 7369 6f6e 002e 676e 752e 7665 .version..gnu.ve
00001110: 7273 696f 6e5f 7200 2e72 656e 612e 6479 rston_f..rela.dy
00001120: 6e00 2e72 656c 612e 706c 740d 2e74 6578 n..rela.plt..tex
00001130: 7400 2e72 6f64 6174 6100 2e65 685f 6672 t..rodata..eh_fr
00001140: 616d 655f 6864 7200 2e65 685f 6672 616d ame_hdr..eh_fram
00001150: 6500 2e70 7265 696e 6974 5f61 7272 6179 e..preinit_array
00001160: 002e 696e 6974 5f61 7272 6179 002e 6669 ..init_array..fi
00001170: 6e69 5f61 7272 6179 002e 6479 6e61 6d69 n1_array..dynam
00001180: 6300 2e64 6174 6100 2e67 6f74 002e 6273 c..data..got..bs
00001190: 7300 2e63 6f6d 6d65 6e74 002e 7269 7363 s..comment..risc
000011a0: 762e 6174 7472 6962 7574 6573 0000 0000 v.attributes....
hello
```


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

```
mp2099@ilab1: ~/cs211/experi x + v
mp2099@ilab1:~/cs211/experiment$ which gcc
/usr/bin/gcc
mp2099@ilab1:~/cs211/experiment$ which x86_64-linux-gnu-gcc
/usr/bin/x86_64-linux-gnu-gcc
mp2099@ilab1:~/cs211/experiment$ which aarch64-linux-gnu-gcc
/usr/bin/aarch64-linux-gnu-gcc
mp2099@ilab1:~/cs211/experiment$ which riscv64-linux-gnu-gcc
/usr/bin/riscv64-linux-gnu-gcc
mp2099@ilab1:~/cs211/experiment$ ls /common/system/riscvi/bin/riscv64-unknown-elf-gcc
/common/system/riscvi/bin/riscv64-unknown-elf-gcc
mp2099@ilab1:~/cs211/experiment$
```

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

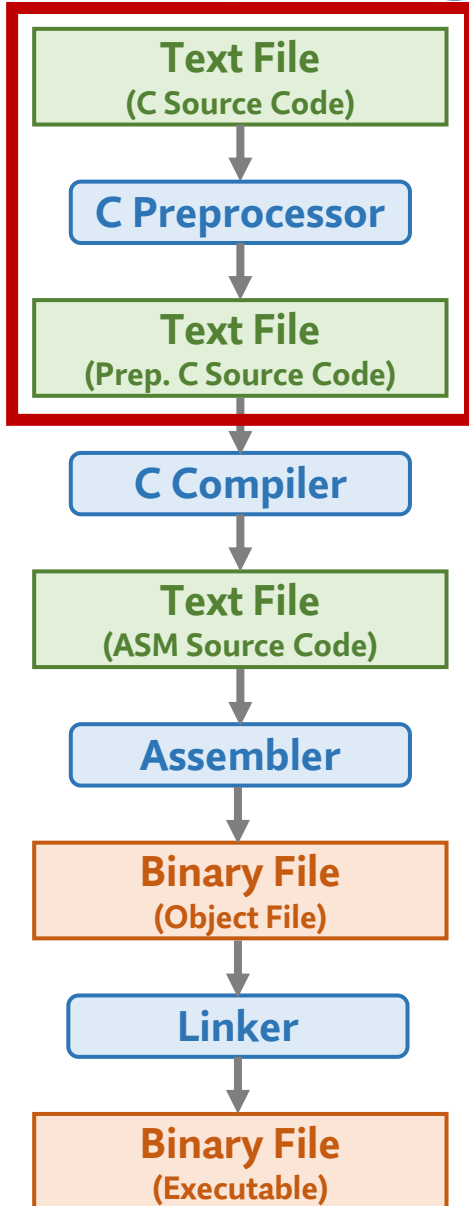
```
netid@ilab2:~$ riscv64-linux-gnu-gcc -Werror [filenames]
```

- 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]
```

Compiling the C Starter Code: Preprocessing



hello_world.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

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

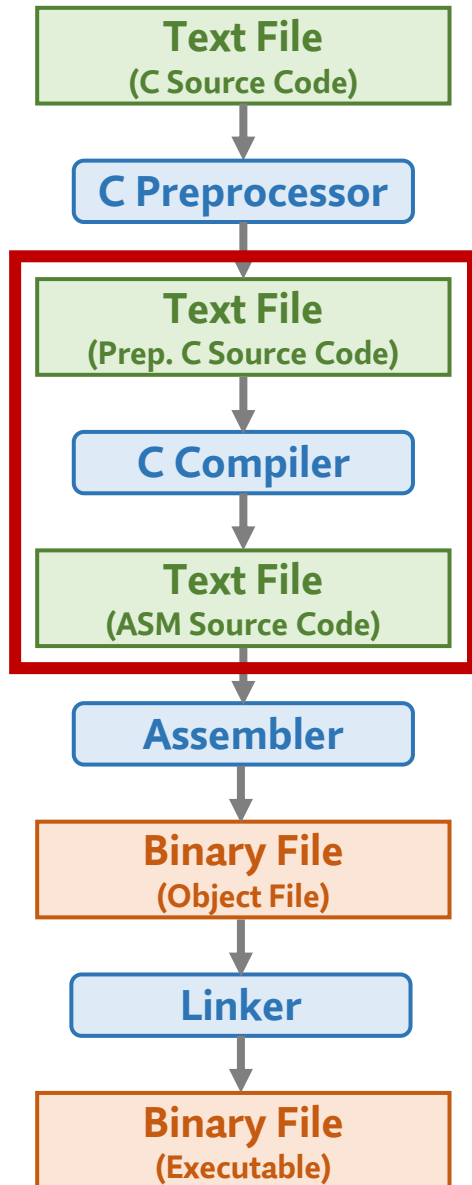
int main(int argc, char *argv[])
{
    puts("Hello World!");
    return 0;
}
```

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;
}
```

C Source Code (Preprocessed)
e.g., ASCII, UTF-8

hello_world.s

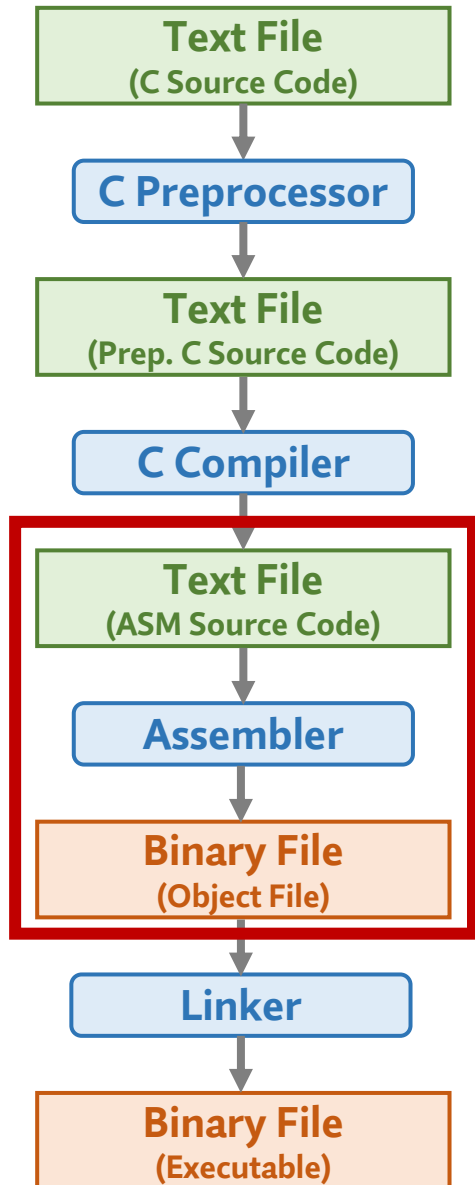
```
.LC0:
.string "Hello World!"
main:
    addi    sp, sp, -32
    sd     ra, 24(sp)
    sd     s0, 16(sp)
    addi   s0, sp, 32
    mv     a5, a0
    sd     a1, -32(s0)
    sw     a5, -20(s0)
    lla    a0, .LC0
    call   puts
    li     a5, 0
    mv     a0, a5
    ld     ra, 24(sp)
    ld     s0,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

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

```
main:  .string "Hello World!"
      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
```

1:1
mapping

hello_world.o

```
mp2099@ilab4: ~/cs211/expe x + v
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
```

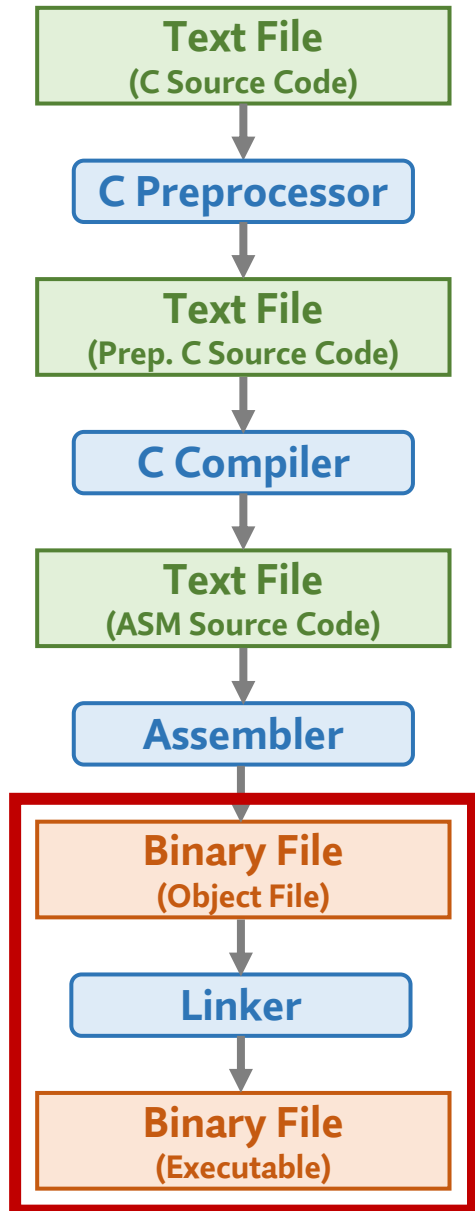
Assembler Source Code
e.g., ASCII, UTF-8

Object File
ELF Format (Binary)

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

```
mp2099@ilab4: ~/cs211/expe x + v
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

```
mp2099@ilab4: ~/cs211/expe x + v
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  .....N.....
00000020: 0000 0000 0000 0000 a003 0000 0000 0000  @.....W.....
00000030: 0500 0000 4000 0000 0000 4000 0c00 0000  ...@.8...@....
00000040: 0111 06cc 22a3 0010 aab7 2330 b4fe 2326  ...p...9.....
00000050: f4fc b707 0000 1305 0700 0700 0000 0700  ..G...Bd.a...
00000060: 0000 0147 3e85 e260 4264 0561 8200 0000  ..G...Bd.a...
00000070: 4805 0c0c 0720 076f 726c 0421 0000 0743  Hello World!..G
00000080: 453a 2020 0720 3134 2e32 2a20 0041 0500  C ( ) 10.2.0.Me
00000090: 0000 7209 7363 7600 015b 0000 0004 1005  ..riscv.[.....
000000a0: 7276 3634 6932 7031 0f6d 3270 305f 6132  rv64imp_m2p0_a2
000000b0: 7031 0f66 3270 320f 0432 7032 0f63 3270  pl_f2p2_d2p2_c2p
000000c0: 305f 7a00 0373 7232 7030 0f7a 0066 0566  @.riscvimp_zifn
```

C Standard Library

```
mp2099@ilab4: ~/cs211/expe x + v
mp2099@ilab4:~/cs211/experiment$ xxd /common/system/riscvi/lib/gcc/r
iscv64-unknown-elf/11.2.0/rv64i/ps4/libgcc.a | head
00000000: 213c 0172 0300 3e0a 2f20 2020 2020 2020  !<arch- /
00000010: 2020 2020 2020 2020 3137 3337 3833 3237  17378327
00000020: 3833 2020 3020 2020 2020 3020 2020 2020  83 0 0
00000030: 3020 2020 2020 2020 3339 3132 2020 2020  0 3912
00000040: 2020 0000 0000 0000 0000 1000 0000 2070  .....P
00000050: 0000 50bc 0000 7fd0 0000 a0bc 0000 c080  ..V.....x..M.
00000060: 0000 e34c 0001 1e78 0001 1e78 0001 4d1c  ...L...x...M.
00000070: 0001 7080 0001 7080 0001 9c0c 0001 bf20  ..p...p.....
00000080: 0001 bf20 0001 e224 0002 0000 0002 0000  ..p...p.....
00000090: 0002 4fcc 0002 a808 0002 a808 0002 cfd0  ..O.....
mp2099@ilab4:~/cs211/experiment$
```

Object Files
ELF Format (Binary)

hello_world

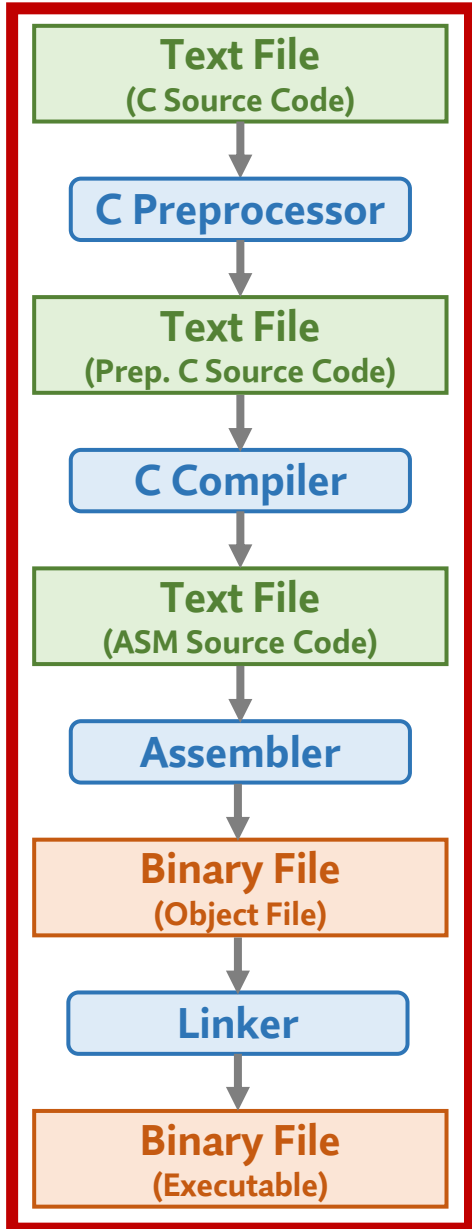
```
mp2099@ilab4: ~/cs211/expe x + v
mp2099@ilab4:~/cs211/experiment$ file hello_world
hello_world: ELF 64-bit LSB executable, UCB RISC-V, RVC, double-flo
at ABI, version 1 (SYSV), statically linked, not stripped
mp2099@ilab4:~/cs211/experiment$ xxd hello_world
00000000: 7f45 4c46 0201 0100 0000 0000 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 0000 0100 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)

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

```
mp2099@ilab4: ~/cs211/expe x + v
mp2099@ilab4:~/cs211/experiment$ /common/system/riscvi/bin/riscv64-unknown-elf-gcc -o hello_world hello_world.o
mp2099@ilab4:~/cs211/experiment$
```

Compiling the C Starter Code: All in One Step



hello_world.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/expe
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/expe
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