

ENGG1003 - Monday Week 12

The C programming language &
version control with Git

Brenton Schulz and Steve Weller

University of Newcastle

24 May 2021

Last compiled: May 23, 2021 8:05pm +10:00

Lecture overview

1 Context

- ▶ ENGG1003
- ▶ what is C?
- ▶ do we even need C?

2 C programming language

- ▶ features and philosophy
- ▶ key language details of C

3 Version control with Git

- ▶ principles
- ▶ practical demonstration (live demo by Brenton)

1) Context

- Recall from last week. . .
- ≤ 2020 , ENGG1003 used *MATLAB* and *C*
 - ▶ **from 2021, ENGG1003 uses Python only**
 - ▶ . . . yet some students will use MATLAB &/or C in later courses
- Thursday week 11: overview of MATLAB
- today's lecture: overview of C

What is C?

- C is a general-purpose, high-level programming language
 - ▶ other high-level languages: Python, MATLAB, Java, C++, FORTRAN, ...
- C has been around since early-1970s and still very popular
 - ▶ IEEE language ranking 2020: #1: Python **#3: C** #10: MATLAB
- C is native language of Linux, Microsoft Windows, OS X, iOS, & Android kernels
- even Python language is written in C
- **C language constructs map efficiently to computer hardware (machine instructions)**

Do we even need C?

- **C is not assessable in ENGG1003**
- BUT... C is currently used in some courses in some Engineering programs:
 - ▶ Aerospace Systems, Computer Systems, Electrical, Mechatronics and Medical Engineering
- key courses which use C language:
 - ▶ ELEC2720, ELEC3730, AERO3600, MCHA3400, MCHA3500
 - ▶ common theme of all these courses is *embedded systems*

Embedded systems

- embedded system: coupled hardware and software designed for a *specific task*
- eg: washing machine
 - ▶ inputs: buttons, water-level, water temperature
 - ▶ output: LED display, motor control
 - ▶ control unit: microprocessor & memory
- *many* other examples: smart TVs, gaming consoles, medical devices, WiFi routers, automotive . . .
- **C language is well-suited to tight coupling of hardware and software in embedded systems**

2) C programming language

Quick recap of *computer language hierarchy*, week 1:

- **machine code**

- ▶ a microprocessor can only understand machine code
- ▶ eg: one instruction for an x86-based processor:
0110 0110 1000 0011 1100 0000 0000 1010
- ▶ very processor-specific!

- **assembly language**

- ▶ human-readable abbreviations (“mnemonics”)
- ▶ eg: machine code
0110 0110 1000 0011 1100 0000 0000 1010
- ▶ same instruction in assembly language
ADD AX, 10
- ▶ very processor-specific

- **high-level languages** eg: C, Python, MATLAB ...

- ▶ human-readable text-based code
- ▶ increased complexity of each high-level instruction
- ▶ eg: machine code
0110 0110 1000 0011 1100 0000 0000 1010
- ▶ same instruction in assembly language
ADD AX, 10
- ▶ same instruction in C
x = x + 10;
- ▶ *not* processor-specific (that's a good thing)

- **compilation process** converts source code (eg: C) to machine code

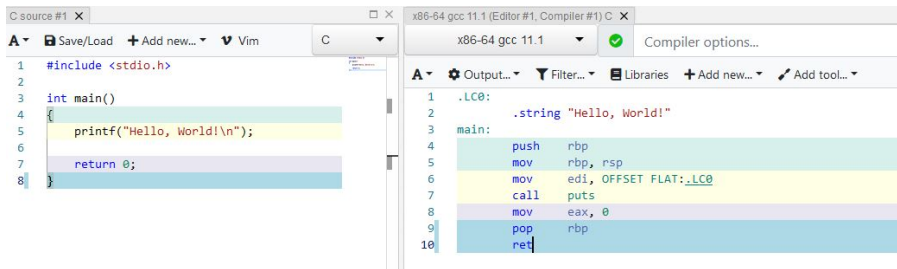
- ▶ source code → pre-processor → compiler → assembler
→ linker → machine code (“executable”)

Hello, World! ...in the C language

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Hello , World!\n");
6     return 0;
7 }
```

- line 1: header file `stdio.h`, like Python `import`
- line 3: function definition for `main()`, returns `int`
- line 5: formatted print, `\n` means “newline”
- line 6: returns “all good” (error-free) message
- lines 4 & 7: `{.}` begin/end of function `main()`

Hello, World! program in C and assembly



The image shows a side-by-side comparison of C code and its assembly translation. The left window, titled 'C source #1', displays a standard C program that prints 'Hello, World!'. The right window, titled 'x86-64 gcc 11.1 (Editor #1, Compiler #1) C', shows the assembly code generated by the compiler. The assembly code includes a label for the string, a main function that pushes the base pointer, moves the stack pointer, calls the 'puts' function with the string address, and then pops the base pointer and returns.

```
C source #1
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Hello, World!\n");
6
7     return 0;
8 }
```

```
x86-64 gcc 11.1
1 .LC0:
2     .string "Hello, World!"
3 main:
4     push    rbp
5     mov     rbp, rsp
6     mov     edi, OFFSET FLAT:._LC0
7     call    puts
8     mov     eax, 0
9     pop     rbp
10    ret
```

- left panel: C program
- right panel: corresponding assembly language code for 64-bit Intel x86
 - ▶ x86 instructions: push, mov, call, pop, ret
- try it yourself at: <https://godbolt.org/>

The C language combines all the power of assembly language
with all the ease-of-use of assembly language.

— *Mark Pearce* —

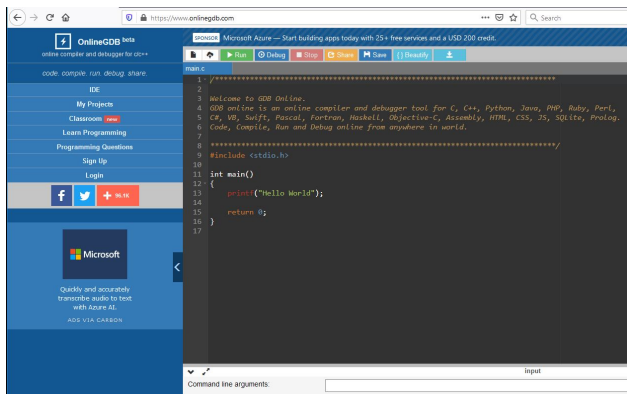
AZ QUOTES



**Learn to think like the computer hates you,
because it does.**

<https://learncodethehardway.org/c/>

OnlineGDB: browser-based C development



edit, compile and run simple C programs with no setup!
<https://www.onlinegdb.com/>

C program: read and print an integer

```
1 #include <stdio.h>
2 int main() {
3     int number;
4
5     printf("Enter an integer: ");
6
7     // reads and stores input
8     scanf("%d", &number);
9
10    // displays output
11    printf("You entered: %d", number);
12
13    return 0;
14 }
```

- line 3: need to “declare” variables before use
- line 8: `scanf()` reads from keyboard, `&number` means “address of variable `number` in memory”

C program: sum of integers using a loop

```
1 #include <stdio.h>
2 int main() {
3     int n, i, sum = 0;
4
5     printf("Enter a positive integer: ");
6     scanf("%d", &n);
7
8     for (i = 1; i <= n; ++i) {
9         sum += i;
10    }
11
12    printf("Sum = %d", sum);
13    return 0;
14 }
```

- lines 8–10: iterate with `for` loop
initialise `i` at 1; continue looping if `i ≤ n` is true;
update `i = i + 1` after each iteration of loop body

C program: power of a number base^{exp}

```
1 #include <stdio.h>
2 int main() {
3     int base, exp;
4     long result = 1;
5     printf("Enter a base number: ");
6     scanf("%d", &base);
7     printf("Enter an exponent: ");
8     scanf("%d", &exp);
9
10    while (exp != 0) {
11        result *= base;
12        --exp;
13    }
14    printf("Answer = %ld", result);
15    return 0;
16 }
```

- lines 10–13: while loop, continues while $\text{exp} \neq 0$
- line 11: $\text{result} = \text{result} * \text{base}$
- line 12: decrement exp , ie: $\text{exp} = \text{exp} - 1$

3) Version control with Git

Programming projects often give rise to two problems:

- 1 programmers want to “roll back” to earlier code
 - ▶ after trying an unsuccessful idea, sometimes faster to load up working code from the past and go from there
 - 2 multiple people contribute to the same project. How should their changes be “merged” into some kind of “master” code listing?
- *version control systems* solve both problems
 - most widely used version control system is *Git*

Apple Inc.: Revision history

[View logs for this page \(view filter log\)](#)

▼ Filter revisions

External tools: [Find addition/removal](#) ([Alternate](#)) · [Find edits by user](#) · [Page statistics](#) · [Pageviews](#)

For any version listed below, click on its date to view it. For more help, see [Help:Page history](#) and

m = minor edit, **→** = section edit, **←** = automatic edit summary

([newest](#) | [oldest](#)) [View](#) ([newer 50](#) | [older 50](#)) ([20](#) | [50](#) | [100](#) | [250](#) | [500](#))

Compare selected revisions

- [\(cur | prev\)](#) ☒ 16:57, 19 May 2021 [Qyedmawfd](#) ([talk](#) | [contribs](#)) .. (361,762 bytes) (+8)
- [\(cur | prev\)](#) ☒ 00:05, 19 May 2021 [Duc4Wikimedia](#) ([talk](#) | [contribs](#)) .. (361,754 bytes) (+)
- [\(cur | prev\)](#) ☐ 14:33, 18 May 2021 [Citation bot](#) ([talk](#) | [contribs](#)) .. (361,420 bytes) (+38) *name changes.* | [Use this bot.](#) | [Report bugs.](#) | [Suggested by Jamesluiz102](#) | [#UCB_webform](#)
- [\(cur | prev\)](#) ☐ 17:59, 17 May 2021 [Tbhotch](#) ([talk](#) | [contribs](#)) .. (361,031 bytes) (+1) .. {
- [\(cur | prev\)](#) ☐ 17:40, 17 May 2021 [JakeyPaul123456](#) ([talk](#) | [contribs](#)) .. (361,030 bytes
- [\(cur | prev\)](#) ☐ 00:17, 17 May 2021 [Tbhotch](#) ([talk](#) | [contribs](#)) .. (361,031 bytes) (0) .. {
- [\(cur | prev\)](#) ☐ 19:51, 16 May 2021 [Wingwatchers](#) ([talk](#) | [contribs](#)) .. (361,031 bytes) (-)
- [\(cur | prev\)](#) ☐ 07:14, 16 May 2021 [Sirius 1098](#) ([talk](#) | [contribs](#)) .. (361,056 bytes) (0) ..
- [\(cur | prev\)](#) ☐ 13:37, 15 May 2021 [Xboxspoon15](#) ([talk](#) | [contribs](#)) .. (361,056 bytes) (0)

- Wikipedia records history of changes
- every time an edit is made, it's recorded in history, even if that change is later reverted
- possible to go back in time to look at that page, at any point in its history

Common Git terms

- **Repository (repo)**

- ▶ store of all the different pieces of the project
- ▶ collection of files + history of changes made to those files

- **Commit**

- ▶ captures a snapshot of the project

- **Push**


- ▶ upload local repository content to a remote repository

- **Pull**

- ▶ update the local version of a repository from a remote

Practical demonstration of Git

- over to you, Brenton ...



	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Next steps

C is not assessable in ENGG1003

- getting started with C, if you need it *for later courses*
- Code::Blocks integrated development environment
 - ▶ <https://www.codeblocks.org/downloads/binaries/>
 - ▶ runs on various platforms:
 - Windows XP / Vista / 7 / 8.x / 10
 - Linux 32- and 64-bit
 - Mac OS X