Curriculum
SE Foundations
Average: **137.49%**

You have a captain's log due before 2024-04-21 (in 1 day)! Log it now!
(/captain_logs/5596018/edit)

0x00. C - Hello, World

C

Weight: 1

Project over - took place from Jul 13, 2023 6:00 AM to Jul 14, 2023 6:00 AM

✓ An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 41.0/41 mandatory & 5.0/12 optional
- **Altogether: 141.67%**
  - Mandatory: 100.0%
  - Optional: 41.67%
  - Calculation: 100.0% + (100.0% * 41.67%) == **141.67%**

Resources

Read or watch:

- Everything you need to know to start with C.pdf (/rltoken/P01aLj9BDfDUOv-y9x82Yw) (*You do not have to learn everything in there yet, but make sure you read it entirely first*)
- Dennis Ritchie (/rltoken/YWFrRob_-Yo-_NQikMLI-g)
- “C” Programming Language: Brian Kernighan (/rltoken/W4oygfMgAp5Hyc7o6QuSYQ)
- Why C Programming Is Awesome (/rltoken/WYdE1novaWa0yt5fzGvLBw)
- Learning to program in C part 1 (/rltoken/aE_pZLbexuLroHA0FmjLbw)
- Learning to program in C part 2 (/rltoken/3a5y1N-0FlTaPbKRxlRLlQ)
- Understanding C program Compilation Process (/rltoken/idYJyVfQRZ9e5aljiT5UKg)
- Betty Coding Style (/rltoken/wJg_qB9ducisfVQNk62htg)
- Hash-bang under the hood (/rltoken/zwv5CHLybXN6KFmsjbu_tg) (*Look at only after you finish consuming the other resources*)
- Linus Torvalds on C vs. C++ (/rltoken/JrokM8Pk6bd9wPqQvEfSAA) (*Look at only after you finish consuming the other resources*)

man or help:

- `gcc`
- `printf (3)`
- `puts`
- `putchar`

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/VGWjGaWZbgcLYTwfLEBmmQ), **without the help of Google**:

General

- Why C programming is awesome

- Who invented C
- Who are Dennis Ritchie, Brian Kernighan and Linus Torvalds
- What happens when you type `gcc main.c`
- What is an entry point
- What is `main`
- How to print text using `printf` , `puts` and `putchar`
- How to get the size of a specific type using the unary operator `sizeof`
- How to compile using `gcc`
- What is the default program name when compiling with `gcc`
- What is the official C coding style and how to check your code with `betty-style`
- How to find the right header to include in your source code when using a standard library function
- How does the `main` function influence the return value of the program

Copyright - Plagiarism

- You are tasked to come up with solutions for the tasks below yourself to meet with the above learning objectives.
- You will not be able to meet the objectives of this or any following project by copying and pasting someone else's work.
- You are not allowed to publish any content of this project.
- Any form of plagiarism is strictly forbidden and will result in removal from the program.

Requirements

C

- Allowed editors: `vi` , `vim` , `emacs`
- All your files will be compiled on Ubuntu 20.04 LTS using `gcc` , using the options `-Wall -Werror -Wextra -pedantic -std=gnu89`
- All your files should end with a new line
- A `README.md` file at the root of the repo, containing a description of the repository
- A `README.md` file, at the root of the folder of *this* project, containing a description of the project
- There should be no errors and no warnings during compilation
- You are not allowed to use `system`
- Your code should use the `Betty` style. It will be checked using betty-style.pl (https://github.com/alx-tools/Betty/blob/master/betty-style.pl) and betty-doc.pl (https://github.com/alx-tools/Betty/blob/master/betty-doc.pl)

Shell Scripts

- Allowed editors: `vi` , `vim` , `emacs`
- All your scripts will be tested on Ubuntu 20.04 LTS
- All your scripts should be exactly two lines long (`$ wc -l file` should print 2)
- All your files should end with a new line
- The first line of all your files should be exactly `#!/bin/bash`

More Info

Betty linter

To run the Betty linter just with command `betty <filename>`:

- Go to the Betty (/rltoken/QkZtBg3ps5iLBlUdX-CPJQ) repository
- Clone the repo (/rltoken/QkZtBg3ps5iLBlUdX-CPJQ) to your local machine
- `cd` into the Betty directory
- Install the linter with `sudo ./install.sh`
- `emacs` or `vi` a new file called `betty`, and copy the script below:

```
#!/bin/bash
# Simply a wrapper script to keep you from having to use betty-style
# and betty-doc separately on every item.
# Originally by Tim Britton (@wintermanc3r), multiargument added by
# Larry Madeo (@hillmonkey)

BIN_PATH="/usr/local/bin"
BETTY_STYLE="betty-style"
BETTY_DOC="betty-doc"

if [ "$#" = "0" ]; then
    echo "No arguments passed."
    exit 1
fi

for argument in "$@" ; do
    echo -e "\n========== $argument =========="
    ${BIN_PATH}/${BETTY_STYLE} "$argument"
    ${BIN_PATH}/${BETTY_DOC} "$argument"
done
```

- Once saved, exit file and change permissions to apply to all users with `chmod a+x betty`
- Move the `betty` file into `/bin/` directory or somewhere else in your `$PATH` with `sudo mv betty /bin/`

You can now type `betty <filename>` to run the Betty linter!

Quiz questions

Great! You've completed the quiz successfully! Keep going! (Show quiz)

Tasks

0. Preprocessor

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a script that runs a C file through the preprocessor and save the result into another file.

- The C file name will be saved in the variable `$CFILE`
- The output should be saved in the file `c`

```
julien@ubuntu:~/c/0x00$ cat main.c
#include <stdio.h>

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    return (0);
}
julien@ubuntu:~/c/0x00$ export CFILE=main.c
julien@ubuntu:~/c/0x00$ ./0-preprocessor
julien@ubuntu:~/c/0x00$ tail c
# 942 "/usr/include/stdio.h" 3 4

# 2 "main.c" 2


# 3 "main.c"
int main(void)
{
 return (0);
}
julien@ubuntu:~/c/0x00$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x00-hello_world`
- File: `0-preprocessor`

Done! Check your code Get a sandbox QA Review

1. Compiler

(/)

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a script that compiles a C file but does not link.

- The C file name will be saved in the variable `$CFILE`
- The output file should be named the same as the C file, but with the extension `.o` instead of `.c`.
    - Example: if the C file is `main.c`, the output file should be `main.o`

```
julien@ubuntu:~/c/0x00$ export CFILE=main.c
julien@ubuntu:~/c/0x00$ cat main.c
#include <stdio.h>

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    return (0);
}
julien@ubuntu:~/c/0x00$ ./1-compiler
julien@ubuntu:~/c/0x00$ ls
0-preprocessor  1-compiler   c            main.o
Makefile               100-intel      main.c  main.s
julien@ubuntu:~/c/0x00$ cat -v main.o | head
^?ELF^B^A^A^@^@^@^@^@^@^@^@^@^@^A^@>^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^P^B^@^@^@^@^@
^@^@^@^@@^@^@^@^@^@@^@^K^@^H^@UHM-^IM-eM-8^@^@^@^@]M-C^@GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.2)
5.4.0 20160609^@^T^@^@^@^@^@^@^@^AzR^@^Ax^P^A^[^L^G^HM-^P^A^@^@^\^@^@^@^\^@^@^@^@^@^@^@^K^@^
@^@^@A^N^PM-^F^BC^M^FF^L^G^H^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^D
^@M-qM-^?^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^C^@^B^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^C^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^C^@^E^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^F^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^C^@^D^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^H^@^@^@^R^@^A^@^@^@^@^@^@^@^@^@^@^K^@^@^
@^@^@^@^@^@main.c^@main^@^@^@^@ ^@^@^@^@^@^@^@^B^@^@^@^B^@^@^@^@^@^@^@^@^@^@^@^@.symtab^@.st
rtab^@.shstrtab^@.text^@.data^@.bss^@.comment^@.note.GNU-stack^@.rela.eh_frame^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^[^@^@^@^A^@^@^@^F^@^@^@^@^@^@^@^@^@^@^@^@^@^@@^@^@^@^@^
@^@^K^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@!^@^@^@^A^@^@^@^C^@^@^@^@
^@^@^@^@^@^@^@^@^@^@K^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@'^@^@^@^H^@^@^@^C^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@@K^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@,^@^@^@^A^@^@^@0^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@K^@
^@^@^@^@^@^@5^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@5^@^@^@^A^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@M-^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@J^@^@^@^A^@^@^@^B^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@M-^@^@^@^@^@^@^@^@8^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^H^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@@E^@^@^@^D^@^@@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@M- ^A^@^@^@^@^@^@^X^@^@^@^@^@^@    ^@^@^@^F^@^@^@^H^@^@^@^@^@^@^X^@^@^@^@^@^@^
@^Q^@^@^@^C^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@M-8^A^@^@^@^@^@^@T^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^B^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@M-8^@^@
^@^@^@^@^@M-X^@^@^@^@^@^@^@
^@^@^@^H^@^@^@^H^@^@^@^@^@^@^@^X^@^@^@^@^@^@^@  ^@^@^@^C^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@M-^P^A^@^@^@^@^@^@^M^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@julien@u
buntu:~/c/0x00$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x00-hello_world`

- File: `1-compiler`

(/)

Done! Check your code Get a sandbox QA Review

2. Assembler

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a script that generates the assembly code of a C code and save it in an output file.

- The C file name will be saved in the variable `$CFILE`
- The output file should be named the same as the C file, but with the extension `.s` instead of `.c`.
    - Example: if the C file is `main.c`, the output file should be `main.s`

(/)

```
julien@ubuntu:~/c/0x00$ export CFILE=main.c
julien@ubuntu:~/c/0x00$ cat main.c
#include <stdio.h>

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    return (0);
}
julien@ubuntu:~/c/0x00$ ./2-assembler
julien@ubuntu:~/c/0x00$ ls
0-preprocessor  1-compiler  2-assembler c  main.c  main.s  Makefile
julien@ubuntu:~/c/0x00$ cat main.s
    .file   "main.c"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    movl    $0, %eax
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.2) 5.4.0 20160609"
    .section    .note.GNU-stack,"",@progbits
julien@ubuntu:~/c/0x00$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x00-hello_world`
- File: `2-assembler`

Done! Check your code Get a sandbox QA Review

3. Name

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a script that compiles a C file and creates an executable named `cisfun`.

- The C file name will be saved in the variable `$CFILE`

```
julien@ubuntu:~/c/0x00$ export CFILE=main.c
julien@ubuntu:~/c/0x00$ cat main.c
#include <stdio.h>

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    return (0);
}
julien@ubuntu:~/c/0x00$ ./3-name
julien@ubuntu:~/c/0x00$ ls
0-preprocessor  1-compiler   3-name  cisfun  main.o  Makefile
100-intel       2-assembler  c       main.c  main.s
julien@ubuntu:~/c/0x00$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x00-hello_world`
- File: `3-name`

Done! Check your code Get a sandbox QA Review

4. Hello, puts

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a C program that prints exactly `"Programming is like building a multilingual puzzle`, followed by a new line.

- Use the function `puts`
- You are not allowed to use `printf`
- Your program should end with the value `0`

```
julien@ubuntu:~/c/0x00$ gcc -Wall -Werror -Wextra -pedantic -std=gnu89 4-puts.c && ./a.out
"Programming is like building a multilingual puzzle
julien@ubuntu:~/c/0x00$ echo $?
0
julien@ubuntu:~/c/0x00$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x00-hello_world`
- File: `4-puts.c`

☑ Done! Check your code Get a sandbox QA Review

5. Hello, printf

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a C program that prints exactly `with proper grammar, but the outcome is a piece of art,`, followed by a new line.

- Use the function `printf`
- You are not allowed to use the function `puts`
- Your program should return `0`
- Your program should compile without warning when using the `-Wall` `gcc` option

```
julien@ubuntu:~/c/0x00$ gcc -Wall -Werror -Wextra -pedantic -std=gnu89 5-printf.c
julien@ubuntu:~/c/0x00$ ./a.out
with proper grammar, but the outcome is a piece of art,
julien@ubuntu:~/c/0x00$ echo $?
0
julien@ubuntu:~/c/0x00$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x00-hello_world`
- File: `5-printf.c`

☑ Done! Check your code Get a sandbox QA Review

6. Size is not grandeur, and territory does not make a nation

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a C program that prints the size of various types on the computer it is compiled and run on.

- You should produce the exact same output as in the example
- Warnings are allowed
- Your program should return `0`
- You might have to install the package `libc6-dev-i386` on your Linux to test the `-m32` `gcc` option

```
julien@ubuntu:~/c/0x00$ gcc 6-size.c -m32 -o size32 2> /tmp/32
julien@ubuntu:~/c/0x00$ gcc 6-size.c -m64 -o size64 2> /tmp/64
julien@ubuntu:~/c/0x00$ ./size32
Size of a char: 1 byte(s)
Size of an int: 4 byte(s)
Size of a long int: 4 byte(s)
Size of a long long int: 8 byte(s)
Size of a float: 4 byte(s)
julien@ubuntu:~/c/0x00$ ./size64
Size of a char: 1 byte(s)
Size of an int: 4 byte(s)
Size of a long int: 8 byte(s)
Size of a long long int: 8 byte(s)
Size of a float: 4 byte(s)
julien@ubuntu:~/c/0x00$ echo $?
0
julien@ubuntu:~/c/0x00$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x00-hello_world`
- File: `6-size.c`

Done! Check your code Get a sandbox QA Review

7. Intel

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Write a script that generates the assembly code (Intel syntax) of a C code and save it in an output file.

- The C file name will be saved in the variable `$CFILE`.
- The output file should be named the same as the C file, but with the extension `.s` instead of `.c`.
  - Example: if the C file is `main.c`, the output file should be `main.s`

(/)

```
julien@ubuntu:~/c/0x00$ export CFILE=main.c
julien@ubuntu:~/c/0x00$ cat main.c
#include <stdio.h>

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    return (0);
}
julien@ubuntu:~/c/0x00$ ./100-intel
julien@ubuntu:~/c/0x00$ cat main.s
    .file   "main.c"
    .intel_syntax noprefix
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    push    rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    mov rbp, rsp
    .cfi_def_cfa_register 6
    mov eax, 0
    pop rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.2) 5.4.0 20160609"
    .section    .note.GNU-stack,"",@progbits
julien@ubuntu:~/c/0x00$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x00-hello_world`
- File: `100-intel`

Done! Check your code Get a sandbox QA Review

8. UNIX is basically a simple operating system, but you have to be a genius to understand the simplicity

#advanced
(/)

Score: 0.0% (*Checks completed: 0.0%*)

Write a C program that prints exactly `and that piece of art is useful" - Dora Korpar, 2015-10-19`, followed by a new line, to the standard error.

- You are not allowed to use any functions listed in the NAME section of the man (3) `printf` or man (3) `puts`
- Your program should return 1
- Your program should compile without any warnings when using the `-Wall` `gcc` option

```
julien@ubuntu:~/c/0x00$ gcc -Wall -Werror -Wextra -pedantic -std=gnu89 -o quote 101-quote.c
julien@ubuntu:~/c/0x00$ ./quote
and that piece of art is useful" - Dora Korpar, 2015-10-19
julien@ubuntu:~/c/0x00$ echo $?
1
julien@ubuntu:~/c/0x00$ ./quote 2> q
julien@ubuntu:~/c/0x00$ cat q
and that piece of art is useful" - Dora Korpar, 2015-10-19
julien@ubuntu:~/c/0x00$ grep printf < 101-quote.c
julien@ubuntu:~/c/0x00$ grep put < 101-quote.c
julien@ubuntu:~/c/0x00$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x00-hello_world`
- File: `101-quote.c`

☐ Done? | Check your code | Ask for a new correction | >_ Get a sandbox | QA Review