0x1C. C - Makefiles

C

- By: Julien Barbier
- Weight: 1
- Project over - took place from Aug 17, 2022 6:00 AM to Aug 18, 2022 6:00 AM
- An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 52.0/52 mandatory & 13.0/13 optional
- **Altogether: 200.0%**
  - Mandatory: 100.0%
  - Optional: 100.0%
  - Calculation: 100.0% + (100.0% * 100.0%) == **200.0%**

```
poofjunior@Brinstar:~/my_arduino_sketch$ 
```

Resources

Read or watch:

- Makefile

Learning Objectives

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

General

- What are `make`, Makefiles
- When, why and how to use Makefiles
- What are rules and how to set and use them
- What are explicit and implicit rules
- What are the most common / useful rules
- What are variables and how to set and use them

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

General

- Allowed editors: `vi`, `vim`, `emacs`
- OS: Ubuntu 20.04 LTS
- Version of `gcc`: 9.3.0
- Version of `make`: GNU Make 4.2.1
- All your files should end with a new line
- A `README.md` file, at the root of the folder of the project, is mandatory

More Info

Files

In the following tasks, we are going to use these files. We want to compile these only.

Tasks

0. make -f 0-Makefile

mandatory

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

Create your first Makefile.

Requirements:

- name of the executable: `school`
- rules: `all`
    - The `all` rule builds your executable
- variables: none

```
julien@ubuntu:~/0x1C. Makefiles$ make -f 0-Makefile

gcc main.c school.c -o school

julien@ubuntu:~/0x1C. Makefiles$ ./school

j#0000000000000000000000000000000000000

j#000000000000000000@Q**g00000000000000

j#0000000000000000*]++]4000000000000000

j#000000000000000k]++]++*N#000000000000

j#0000000000000*C+++]++]++]J*0000000000

j#00000000000@+]++qwwwp=]++++]*00000000

j#0000000000*+++]q#0000k+]+]++]4#000000

j#00000000*C+]+]w#0000*]+++]+]++0000000

j#0000we+]wW000***C++]++]+]++++40000000

j#000000000*C+]+]]+]++]++]++]+q#0000000

j#0000000*]+]+++++++]++]+++]+++J0000000

j#000000C++]=]+]+]+]++]++]+]+]+]=000000

j#00000k+]++]+++]+]++qwW0000000AgW00000

j#00000k++]++]+]+++qW#00000000000000000

j#00000A]++]++]++]++J**0000000000000000

j#000000e]++]+++]++]++]J000000000000000

j#0000000A]++]+]++]++]++000000000000000

j#000000000w]++]+]++]+qW#00000000000000

j#00000000000w]++++]*0##000000000000000

j#0000000000000Ag]+]++*0000000000000000

j#00000000000000000we]+]Q00000000000000

j#0000000000000@@+wgdA]+J00000000000000

j#0000000000000k?qwgdC=]4#0000000000000
```

```
j#00000000000000w]+]++qw#00000000000000
"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
julien@ubuntu:~/0x1C. Makefiles$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1C-makefiles`
- File: `0-Makefile`

Done! Help Check your code Get a sandbox QA Review

1. make -f 1-Makefile

mandatory

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

Requirements:

- name of the executable: `school`
- rules: `all`
  - The `all` rule builds your executable
- variables: `CC`, `SRC`
  - `CC`: the compiler to be used
  - `SRC`: the `.c` files

```
julien@ubuntu:~/0x1C. Makefiles$ make -f 1-Makefile
gcc main.c school.c -o school
julien@ubuntu:~/0x1C. Makefiles$ make -f 1-Makefile
gcc main.c school.c -o school
julien@ubuntu:~/0x1C. Makefiles$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1C-makefiles`
- File: `1-Makefile`

Done! Help Check your code Get a sandbox QA Review

2. make -f 2-Makefile

mandatory

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

Create your first useful Makefile.

Requirements:

- name of the executable: `school`
- rules: `all`
  - The `all` rule builds your executable
- variables: `CC`, `SRC`, `OBJ`, `NAME`
  - `CC`: the compiler to be used
  - `SRC`: the `.c` files
  - `OBJ`: the `.o` files
  - `NAME`: the name of the executable
- The `all` rule should recompile only the updated source files
- You are not allowed to have a list of all the `.o` files

```
julien@ubuntu:~/0x1C. Makefiles$ make -f 2-Makefile
gcc    -c -o main.o main.c
gcc    -c -o school.o school.c
gcc main.o school.o -o school
julien@ubuntu:~/0x1C. Makefiles$ make -f 2-Makefile
gcc main.o school.o -o school
julien@ubuntu:~/0x1C. Makefiles$ echo "/* School */" >> main.c
julien@ubuntu:~/0x1C. Makefiles$ make -f 2-Makefile
gcc    -c -o main.o main.c
gcc main.o school.o -o school
julien@ubuntu:~/0x1C. Makefiles$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1C-makefiles`
- File: `2-Makefile`

Done! Help Check your code Get a sandbox QA Review

3. make -f 3-Makefile

mandatory

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

Requirements:

- name of the executable: `school`
- rules: `all`, `clean`, `oclean`, `fclean`, `re`
  - `all`: builds your executable
  - `clean`: deletes all Emacs and Vim temporary files along with the executable

    - `oclean`: deletes the object files
    - `fclean`: deletes all Emacs and Vim temporary files, the executable, and the object files
    - `re`: forces recompilation of all source files
- variables: `CC`, `SRC`, `OBJ`, `NAME`, `RM`
    - `CC`: the compiler to be used
    - `SRC`: the `.c` files
    - `OBJ`: the `.o` files
    - `NAME`: the name of the executable
    - `RM`: the program to delete files
- The `all` rule should recompile only the updated source files
- The `clean`, `oclean`, `fclean`, `re` rules should never fail
- You are not allowed to have a list of all the `.o` files

```
julien@ubuntu:~//0x1C. Makefiles$ ls -1

0-Makefile

1-Makefile

2-Makefile

3-Makefile

school.c

main.c

main.c~

m.h

julien@ubuntu:~/0x1C. Makefiles$ make -f 3-Makefile

gcc    -c -o main.o main.c

gcc    -c -o school.o school.c

gcc main.o school.o -o school

julien@ubuntu:~/0x1C. Makefiles$ make all -f 3-Makefile

gcc main.o school.o -o school

julien@ubuntu:~/0x1C. Makefiles$ ls -1

0-Makefile

1-Makefile

2-Makefile

3-Makefile

school

school.c

school.o

main.c
main.c~
main.o
m.h
julien@ubuntu:~/0x1C. Makefiles$ make clean -f 3-Makefile
rm -f *~ school
julien@ubuntu:~/0x1C. Makefiles$ make oclean -f 3-Makefile
rm -f main.o school.o
julien@ubuntu:~/0x1C. Makefiles$ make fclean -f 3-Makefile
rm -f *~ school
rm -f main.o school.o
julien@ubuntu:~/0x1C. Makefiles$ make all -f 3-Makefile
gcc    -c -o main.o main.c
gcc    -c -o school.o school.c
gcc main.o school.o -o school
julien@ubuntu:~/0x1C. Makefiles$ make all -f 3-Makefile
gcc main.o school.o -o school
julien@ubuntu:~/0x1C. Makefiles$ make re -f 3-Makefile
rm -f main.o school.o
gcc    -c -o main.o main.c
gcc    -c -o school.o school.c
gcc main.o school.o -o school
julien@ubuntu:~/0x1C. Makefiles$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1C-makefiles`
- File: `3-Makefile`

Done! Help Check your code Get a sandbox QA Review

4. A complete Makefile

mandatory

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

Requirements:

- name of the executable: `school`

- rules: `all`, `clean`, `fclean`, `oclean`, `re`
    - `all`: builds your executable
    - `clean`: deletes all Emacs and Vim temporary files along with the executable
    - `oclean`: deletes the object files
    - `fclean`: deletes all Emacs and Vim temporary files, the executable, and the object files
    - `re`: forces recompilation of all source files
- variables: `CC`, `SRC`, `OBJ`, `NAME`, `RM`, `CFLAGS`
    - `CC`: the compiler to be used
    - `SRC`: the `.c` files
    - `OBJ`: the `.o` files
    - `NAME`: the name of the executable
    - `RM`: the program to delete files
    - `CFLAGS`: your favorite compiler flags: `-Wall -Werror -Wextra -pedantic`
- The `all` rule should recompile only the updated source files
- The `clean`, `oclean`, `fclean`, `re` rules should never fail
- You are not allowed to have a list of all the `.o` files

```
julien@ubuntu:~/0x1C. Makefiles$ make all -f 4-Makefile
gcc -Wall -Werror -Wextra -pedantic   -c -o main.o main.c
gcc -Wall -Werror -Wextra -pedantic   -c -o school.o school.c
gcc main.o school.o -o school
julien@ubuntu:~/0x1C. Makefiles$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1C-makefiles`
- File: `4-Makefile`

Done! Help Check your code Get a sandbox QA Review

5. Island Perimeter

mandatory

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

Technical interview preparation:

- You are not allowed to google anything
- Whiteboard first

Create a function `def island_perimeter(grid):` that returns the perimeter of the island described in `grid`:

- `grid` is a list of list of integers:
    - 0 represents a water zone

    - 1 represents a land zone
    - One cell is a square with side length 1
    - Grid cells are connected horizontally/vertically (not diagonally).
    - Grid is rectangular, width and height don't exceed 100
- Grid is completely surrounded by water, and there is one island (or nothing).
- The island doesn't have "lakes" (water inside that isn't connected to the water around the island).

Requirements:

- First line contains `#!/usr/bin/python3`
- You are not allowed to import any module
- Module and function must be documented

```
guillaume@ubuntu:~/0x1C$ cat 5-main.py
#!/usr/bin/python3
"""
5-main
"""
island_perimeter = __import__('5-island_perimeter').island_perimeter

if __name__ == "__main__":
    grid = [
        [0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0],
        [0, 0, 0, 0, 0, 0]
    ]
    print(island_perimeter(grid))

guillaume@ubuntu:~/0x1C$
guillaume@ubuntu:~/0x1C$ ./5-main.py
12
guillaume@ubuntu:~/0x1C$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1C-makefiles`
- File: `5-island_perimeter.py`

Done! Help Check your code Get a sandbox QA Review

6. make -f 100-Makefile

#advanced

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

Requirements:

- name of the executable: `school`
- rules: `all`, `clean`, `fclean`, `oclean`, `re`
  - `all`: builds your executable
  - `clean`: deletes all Emacs and Vim temporary files along with the executable
  - `oclean`: deletes the object files
  - `fclean`: deletes all Emacs and Vim temporary files, the executable, and the object files
  - `re`: forces recompilation of all source files
- variables: `CC`, `SRC`, `OBJ`, `NAME`, `RM`, `CFLAGS`
  - `CC`: the compiler to be used
  - `SRC`: the `.c` files
  - `OBJ`: the `.o` files
  - `NAME`: the name of the executable
  - `RM`: the program to delete files
  - `CFLAGS`: your favorite compiler flags: `-Wall -Werror -Wextra -pedantic`
- The `all` rule should recompile only the updated source files
- The `clean`, `oclean`, `fclean`, `re` rules should never fail
- You are not allowed to have a list of all the `.o` files
- You have to use `$(RM)` for the cleaning up rules, but you are not allowed to set the `RM` variable
- You are not allowed to use the string `$(CC)` more than once in your Makefile
- You are only allowed to use the string `$(RM)` twice in your Makefile
- You are not allowed to use the string `$(CFLAGS)` (but the compiler should still use the flags you set in this variable)
- You are not allowed to have an `$(OBJ)` rule
- You are not allowed to use the `%.o: %.c` rule
- Your Makefile should work even if there is a file in the folder that has the same name as one of your rule
- Your Makefile should not compile if the header file `m.h` is missing

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1C-makefiles`
- File: `100-Makefile`

Done! Help Check your code Get a sandbox QA Review