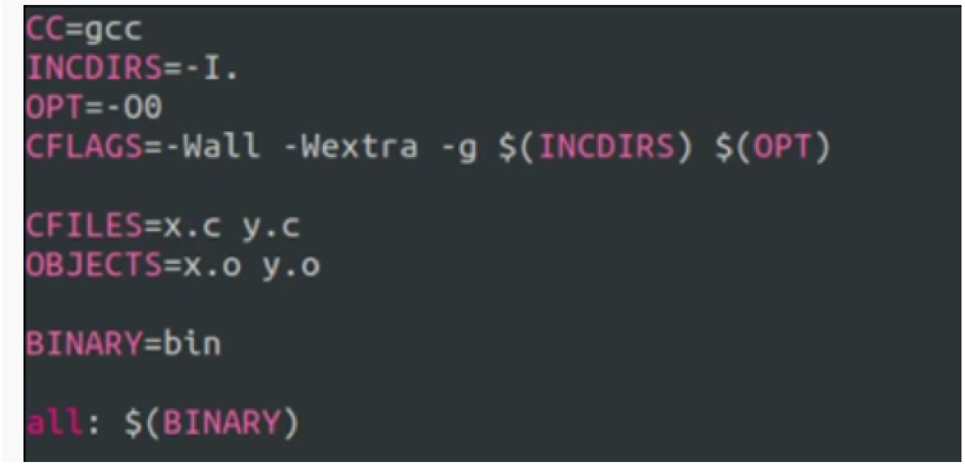
Makefiles

- ## Video: https://www.youtube.com/watch?v=_r7i5X0rXJk

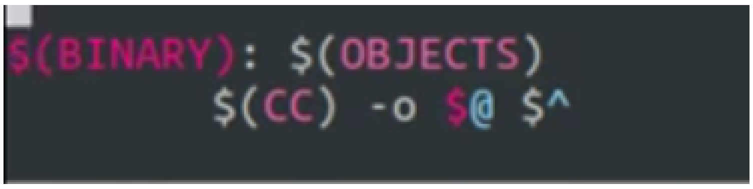
- A Makefile helps to compile all of the files in a project instead of using the command with all of the names in it
  - Command <make> will execute the Makefile
  - Command <make clean> will execute the section "clean" inside the Makefile
    - in the clean section files will be defined which are going to be deleted when "cean" is executed
    - Syntax:
      - clean:
        rm *.o filename   <- This will delete alls object-files and the file called"filename"
  - Syntax of a makefile: (NOTE: the "action" comes after a new line and a tab
    - *target: dependencies*
      *   action*
    - *main.o: main.c*   ***<- create main.o ("o" means object) every time that main.c changes***
      gcc main.c -o main   ***<- compile main.c with the gcc compiler and create executable named "main"***

Video: https://www.youtube.com/watch?v=DtGrdB8wQ_8

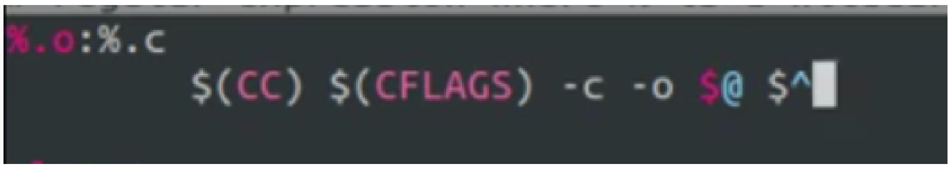
- A Makefile will only recompile the files that have changed or the one that have a dependencies on the changed file
- Variables in the Makefile:
  - see screenshots below the variablenames in pink and all caps
  - to unse variables write "$(variablename)"
- Command <make all>   **<- This is the same als <make> because make will execute the first rule, which should be "all"**
  - the rule for "all" sould be on the first level in order to make sure that all the files inside the makefile will be created

- 

```
CC=gcc
INCDIRS=-I.
OPT=-O0
CFLAGS=-Wall -Wextra -g $(INCDIRS) $(OPT)

CFILES=x.c y.c
OBJECTS=x.o y.o

BINARY=bin

all: $(BINARY)
```

- Command <make name>
  - it's a rule bases on a variable $(NAME)
    - 

```
$(BINARY): $(OBJECTS)
        $(CC) -o $@ $^
```

- The sign "%" is a wildcard / placeholder for anything:
  - %.c   <- This would be any .c file
- The sign "$@" means take the string on the left side of the ":"   **<- int this case the name we want go give our file after compiling**
- The sign "$^" means take the string on the right side of the ":"   **<- in this case the name of the .c-file**
  - 

```
%.o:%.c
        $(CC) $(CFLAGS) -c -o $@ $^
```

Documentation: https://epitech-2022-technical-documentation.readthedocs.io/en/latest/makefiles.html

---

- ### Generic Makefile

  A simple Makefile is composed of your project source files (the .c files) and some `rules` to make your `Make` command work.

  You have to list your source files like this:

  SRC = ./main.c
  ./file.c

  After that, you can use them to build your objects. It will take all `.c` files in `$(SRC)` and compile them into `.o` files.

  OBJ = $(SRC:.c=.o)

For the compilation there is a feature that allow you to compile each `.c` with flags, it's the `+=`. For example let's add verification of errors flags : `-Werror -Wextra` and a flags to find `.h` of your project : `-I./include`. You can call this variable `CFLAGS` for compilation's flags.

CFLAGS += -Werror -Wextra -I./include

Be careful !

You don't have to call this variable in your Makefile, he will solo add it to the compilation of your `.c`.

Now, set the name of your final binary using `NAME`, so the AutoGrader can find your binary correctly.

NAME = binary_name

Then, it is mandatory to create a $(NAME) rule that will execute other rules, and render a binary.

$(NAME): $(OBJ)

gcc -o $(NAME) $(OBJ)

all: $(NAME)

Pro-tip

When you have a rule like `$(NAME)`, the rules put in the same line will be used as mandatory rules. After those rules have been called, the command on the next lines will be called.

For instance, this will execute the `ls` command without executing any previous rule.

list:

ls

You can also have some rules that permit you to clean-up your folder without having to do it manually.

For instance, this `clean` will remove `.o` files. Also, `fclean` will remove `.o` files and the binary. `re` will do `fclean` and re-make your binary.

clean:

rm -f $(OBJ)

fclean: clean

rm -f $(NAME)

re: fclean all