```
  GNU nano 7.2                          Makefile
TARGET=hello
CC=gcc
OUTDIR=build
FILENAME=hello.c
FILENAME_O=hello.o
NEW_FILENAME=src/hello.c
all: ${TARGET}
${TARGET}: ${FILENAME_O} ${OUTDIR}
        ${CC} ${OUTDIR}/${FILENAME_O} -o ${OUTDIR}/${TARGET}
        cp ${NEW_FILENAME} ${OUTDIR}/${FILENAME}
${OUTDIR}:
        mkdir ${OUTDIR}
${FILENAME_O}: ${NEW_FILENAME} ${OUTDIR}
        ${CC} -c ${NEW_FILENAME} -o ${OUTDIR}/${FILENAME_O}
clean:
        rm -rf ${OUTDIR}
```

TARGET=hello:

to jest nazwa pliku wynikowego. Po kompilacji musi powstać plik `hello`.

CC=gcc:

definiujemy, jakiego kompilatora chcemy użyć; `gcc` (GNU Compiler Collection) to kompilator języka C.

FILENAME=hello.c

nazwa głównego pliku z kodem w języku C.

```c
#include <stdio.h>

int main() {
        printf("Hello, World!\n");

        return 0;
}
```

FILENAME_O=hello.o

nazwa pliku obiektowego, który powstanie po kompilacji (- c).

NEW_FILENAME=src/hello.c

ta linijka określa ścieżkę do pliku źródłowego hello.c, który znajduje się w katalogu src. Dzięki użyciu zmiennej **NEW_FILENAME** możemy łatwo zmienić lokalizację pliku źródłowego bez konieczności modyfikowania całego Makefile'a.
(np. jeśli plik hello.c zostałby przeniesiony do innego folderu, wystarczyłoby zmienić wartość tej jednej zmiennej (NEW_FILENAME=nowy_dir/hello.c), a pozostała część Makefile'a nadal działałaby poprawnie.

all: ${TARGET}

reguła **all** to domyślny cel.
kiedy wpisujemy po prostu "make" w terminalu, to make wykona właśnie regułę **all**.
all zależy od **TARGET**, więc aby wykonać regułę **all**, najpierw musi zostać zbudowany ${TARGET}.

${TARGET}: ${FILENAME_O} ${OUTDIR}

ta linijka mówi, że program wynikowy (hello) zależy od:

- pliku obiektowego ${FILENAME_O} (czyli hello.o),
- katalogu ${OUTDIR} (build).

Jeśli któryś z tych elementów nie istnieje, make wykona komendy poniżej:

```
${FILENAME_O}: ${NEW_FILENAME} ${OUTDIR}
```

Tutaj definiujemy, jak stworzyć plik obiektowy hello.o.
Zależności: plik źródłowy (src/hello.c) oraz katalog (build).

${CC} -c ${NEW_FILENAME} -o ${OUTDIR}/${FILENAME_O}

${CC}: za pomocą kompilatora gcc powstaje gotowy program z pliku obiektowego

-c: kompiluj, ale nie linkuj (tworzy plik .o)

-o: → zapisz wynik do folderu (build)

Czyli: z pliku src/hello.c powstanie build/hello.o.

${OUTDIR}:

mkdir ${OUTDIR}: tworzymy katalog build, jeśli go nie ma.

zawartość ${TARGET}: ${FILENAME_O} ${OUTDIR}:

${CC} ${OUTDIR}/${FILENAME_O} -o ${OUTDIR}/${TARGET}: kompilacja programu wynikowego

-o oznacza „output”, czyli wynik zapisz w ${OUTDIR}/${TARGET} (czyli np. build/hello)

cp ${NEW_FILENAME} ${OUTDIR}/${FILENAME}: kopiowanie pliku źródłowego

kopiuje oryginalny plik źródłowy (src/hello.c) do folderu build(żeby tam był cały projekt skompilowany razem).

clean: służy do usuwania wyników kompilacji

rm -rf ${OUTDIR}: usuwa cały katalog build wraz z zawartością.

WYNIK

```
karyna@karyna-VirtualBox:~/lab2$ make
mkdir build
gcc -c src/hello.c -o build/hello.o
gcc build/hello.o -o build/hello
cp src/hello.c build/hello.c
karyna@karyna-VirtualBox:~/lab2$ ls
build  hello  Makefile  src
karyna@karyna-VirtualBox:~/lab2$ make clean
rm -rf build
karyna@karyna-VirtualBox:~/lab2$
```

```
karyna@karyna-VirtualBox:~/lab2/build$ ./hello
Hello, World!
```

Aby sprawdzić poprawność działania pliku **Makefile**, uruchomiłam polecenia:

make

make clean

make

```
karyna@karyna-VirtualBox:~/lab2$ make
mkdir build
gcc -c src/hello.c -o build/hello.o
gcc build/hello.o -o build/hello
cp src/hello.c build/hello.c
karyna@karyna-VirtualBox:~/lab2$ make clean
rm -rf build
karyna@karyna-VirtualBox:~/lab2$ make
mkdir build
gcc -c src/hello.c -o build/hello.o
gcc build/hello.o -o build/hello
cp src/hello.c build/hello.c
```

Po wykonaniu komendy make clean wszystkie pliki utworzone podczas kompilacji zostały usunięte, a katalog projektu został przywrócony do stanu początkowego.
Następne wywołanie make ponownie skompilowało cały program **bez żadnych błędów**, co potwierdza, że plik **Makefile** został przygotowany prawidłowo i zawiera wszystkie niezbędne reguły oraz zależności.