# CSC209H Worksheet: Makefiles

In this worksheet we will examine the `Makefile` for Assignment 2 for a previous offering of the course. Remember that the purpose of `make` is to automate the build process so that

- we don't have to type a long compile command every time we want to compile our code, and
- dependencies between files are tracked and source files are only recompiled when necessary.

1. Before we look at the full `Makefile`, consider the following `Makefile` rule:

a
```
test_print: test_print.o ptree.o
        gcc -Wall -g -o test_print test_print.o ptree.o
```

   (a) Circle the target.

   (b) Underline the prerequisites. What is another term for prerequisites? Dependencies

   (c) How many actions does this rule have? 1

   (d) What does a file that ends in `.o` contain? How is it generated?

   A .o file is an object file that contains compiled code with placeholders for links to object code in other files or libraries. It is generated using the -c option for gcc

2. The remaining questions are about the `Makefile` on the other side of this page.

   Suppose that the only files in the current working directory are the source files, the header files, and the `Makefile`. In other words, this is the first time any compilation happens.

   (a) If we were to run `make test_load_data` which rule is evaluated first?

   test_load_data (We will name rules by their target)

   (b) What new files would be created?

   dectree.o , test_load_data.o, test_load_data

   (c) What is the *last* action that is executed in the `make` command above?

   gcc -Wall -g -o test_load_data test_load_data.o dectree.o -lm

   (d) Which files will the pattern rule (`%.o : %.c`) match on?

   dectree.c classifier.c (and their
   test_load_data.c corresponding .o files)

   (e) If we the modify `dectree.c` and run `make test_load_data` again, which rules are evaluated? Which actions are executed?

   rules: test_load_data
   dec_tree.o
   test_load_data.o

   → gcc -Wall -g -c dec_tree.c
   → gcc -Wall -g -o test_load_data test_load_data.o dectree.o -lm

# CSC209H Worksheet: Makefiles

Here are two versions of the same Makefile: the first doesn't use the special variables or pattern rule:

```
FLAGS = -Wall -g
.PHONY: clean all

all: classifier  test_load_data

classifier : dectree.o classifier.o
    gcc ${FLAGS} -o classifier dectree.o classifier.o -lm

test_load_data : dectree.o test_load_data.o
    gcc ${FLAGS} -o test_load_data dectree.o test_load_data.o -lm

classifier.o : classifier.c dectree.h
    gcc ${FLAGS} -c classifier.c

test_load_data.o : test_load_data.c dectree.h
    gcc ${FLAGS} -c test_load_data.c

dectree.o : dectree.c dectree.h
    gcc ${FLAGS} -c dectree.c

datasets : datasets.tgz
    tar xzf datasets.tgz

clean:
    rm classifier test_load_data *.o
```

The second Makefile is shorter and arguably easier to add to and modify because it uses the variables.

```
FLAGS = -Wall -g
.PHONY: clean all

all: classifier  test_load_data

classifier : dectree.o classifier.o
    gcc ${FLAGS} -o $@ $^ -lm

test_load_data : dectree.o test_load_data.o
    gcc ${FLAGS} -o $@ $^ -lm

classifier.o : dectree.h
dectree.o : dectree.h
test_load_data.o : dectree.h

%.o : %.c
    gcc ${FLAGS} -c $<

datasets : datasets.tgz
    tar xzf datasets.tgz

clean:
    rm classifier test_load_data *.o
```

**Makefile syntax**

| Variable | Meaning |
|---|---|
| `$@` | Target |
| `$<` | First prerequisite |
| `$?` | All out of date prerequisites |
| `$^` | All prerequisites |