02635 Fall 2016 — Module 4

Homework

- Read chapter 7 pp. 287-298 and chapter 8 in "Beginning C"
- Read section 8.3 (pp. 92-96), sections 14.1-14.2 (pp. 195-198), chapter 15, and section 17.1 (pp. 223-226) in "Writing Scientific Software"

Makefiles

Suppose you have written a C program with a single source file `myprogram.c`. To compile your program using `make`, create a file with the name `Makefile` (without an extension) with the following content:

```
myprogram: myprogram.c
    cc -Wall myprogram.c -o myprogram
```

You may now use `make` to compile your program:

```
$ make myprogram
```

Alternatively, if you decide to use a different name for your makefile than the standard name `Makefile`, you will need to use the `-f` option to tell `make` which file to use:

```
$ make -f MyMakefile myprogram
```

Now suppose that you have written a C program that consists of a header file `myheader.h` and two source files `myprogram.c` and `routine1.c`. To compile your program using `make`, create a makefile with the following content:

```
# Rule for linking
myprogram: myprogram.o routine1.o
    cc -o myprogram myprogram.o routine1.o

# Rule for generating object code myprogram.o
myprogram.o: myprogram.c myheader.h
    cc -c myprogram.c -Wall

# Rule for generating object code routine1.o
routine1.o: routine1.c
    cc -c routine1.c -Wall
```

Now execute the command

```
$ make myprogram
```

to compile your program.

Makefile template

The last example shows that the naive way of writing a makefile for a large project with many source files can be tedious. Luckily `make` allows us to define variables and rules, and this makes it possible to write a generic makefile that can be used in many situations. Below is a basic template that you can modify and use for the remaining exercises and assignments in this course. Note that the lines that start with the hashtag character `#` are comments, and hence ignored by `make`.

```makefile
# DTU 02635 Makefile template

# C compiler (cc, gcc, clang, ...)
CC=cc

# Source files
SRCS=file1.c file2.c file3.c

# Name of executable
EXECUTABLE=myprog

# Compiler flags
CFLAGS=-g -Wall -Wextra

# Linker flags
# Example: LDFLAGS=-L/path/to/library
LFLAGS=

# External libraries
# Example: LIBS=-lm
LIBS=-lm

# Header directories to include
# Example: INCLUDES=-I/path/to/headers/
INCLUDES=

OBJS=$(SRCS:.c=.o)

all: $(SRCS) $(EXECUTABLE)

$(EXECUTABLE): $(OBJS)
    $(CC) $(OBJS) $(LFLAGS) $(LIBS) -o $@

%.o: %.c
    $(CC) -c $(CFLAGS) $(INCLUDES) $< -o $@

clean:
    rm *.o $(EXECUTABLE)

run: $(EXECUTABLE)
    ./$<
```

Exercises

1. Do exercises 7-1 and 7-4 in "Beginning C"
2. Do exercise 8-1 in "Beginning C"
    - Hint: Reuse your code for exercise 7-1.
    - Optional: Write a makefile for your code.

3. Take this quiz to test your understanding of **functions**
4. Catch up on unfinished exercises from previous weeks.

Optional exercise:
Rewrite your code from Part II of the module 3 exercises using what you have learned from chapter 8 in "Beginning C". Your code should - be modular and consist of components/functions that are easy to test; - use dynamic memory allocation for arrays.