Makefile

In order to make our source code executable (a file that we can run) we need first to compile all the c files and then link the compilation products to an executable file. If we have external libraries we need to link these libraries as well. We can do this process manually through the console, but it might take a lot of time when our project is bigger than 2 files. Alternatively, we can use a special file, the makefile, which contains all the data and commands needed to transform the entire source code files to an executable program. Let's write a makefile to the hello world project.\\ Assume our project contains the following files: HelloWorld.c, HelloWorld.h and Run.c In this course you are required to use the flags used here.

Reminder: Compile your C project manually

To compile hello world....

1. Open a terminal.
2. Go to your project's root, e.g.`cd ~/splab/labA`
3. Clean:`rm -f *.o hello`
4. Compile: `gcc -g -m32 -Wall -c -o hello.o HelloWorld.c`
5. Compile:`gcc -g -m32 -Wall -c -o run.o Run.c`
6. Link: `gcc -g -m32 -Wall -o hello hello.o run.o`
7. Notice we do not compile a Header file.

gcc parameters

-o *fileName* Place output in file *fileName*; creates by default a file called: a.out

-c Compile or assemble the source files, but do not link. The compiler output is an object file corresponding to each source file.

-L*dir* Add directory *dir* to the list of directories to be searched for **-l**.

-l*lib* Use the library named *lib* when linking. (C programs often require **gcc -l** for successful linking.)

-O Optimize. Use this to yield optimized code. (for example that runs faster).

-g Add debug information.

-W What warning level would you like to receive.

-m32 Specifies the output file format to be 32bit.

Compiling manually a C project with Assembly files

1. To compile hello world assuming we have HelloWorld.c, HelloWorld.h and start.s Open a terminal.
2. Go to your project's root, e.g.`cd ~/splab/labA`
3. Clean:`rm -f *.o hello`
4. Compile: `gcc -m32 -g -Wall -c -o hello.o HelloWorld.c`
5. Compile: `nasm -g -f elf -w+all -o start.o start.s`
6. Link: `gcc -m32 -g -Wall -o hello hello.o start.o`

Note that the only difference here is that assembly language files are "compiled" by the assembler, rather than the C compiler gcc. Linking object files created by the assembler is the same as for objects generated by the C compiler.

nasm parameters

-o *fileName* Place output in file *fileName*; creates by default a file called: [original filename].o

-f *format* Specifies the output file format. To see a list of valid output formats, use the -hf option

-g Add debug information.

-w+all Warning level we want to receive.

Makefiles

Makefiles (and the program which uses them) are used to specify dependencies between abstract *targets*, and what needs to be done to achieve each target. More specifically, a Makefile is a text file containing a description of our targets (the most common case of a target is an object), followed by their dependencies and next by the actions which are necessary to build these targets. Target names and dependencies are assumed to be **files**. Consider a simple target, T_1, which depends on D_1 and D_2 (all of which are files). To build T_1, the make utility will work as follows.

1. If either D_1 or D_2 do not exist, build them (recursively).
2. Check if both D_1 and D_2 are up to date. If not, build them recursively.
3. Check the date of T_1 (the modification time). If T_1 is at least as new as BOTH D_1 and D_2, we are done, as T_1 is up to date.
4. Otherwise, follow the instructions given to build T_1.

When you type make in your shell, the script will look for a file called "makefile" in the same directory and will execute it, using the rules defined in the "makefile" file. By default, make will only execute the **first** target in the makefile. So, make sure the first target causes a complete build.

C project makefile

```
#format is target-name: target dependencies


#{-tab-}actions
 # All Targets
all: hello
 # Tool invocations
# Executable "hello" depends on the files hello.o and run.o.
hello: hello.o Run.o
	gcc -g -m32 -Wall -o hello hello.o Run.o
 # Depends on the source and header files
hello.o: HelloWorld.c HelloWorld.h
	gcc -m32 -g -Wall -c -o hello.o HelloWorld.c
Run.o: Run.c
	gcc -m32 -g -Wall -c -o Run.o Run.c
 #tell make that "clean" is not a file name!
.PHONY: clean
 #Clean the build directory
clean:
	rm -f *.o hello
```

C project with Assembly makefile

```
#format is target-name: target dependencies


#{-tab-}actions
```

```
 # All Targets

all: hello

 # Tool invocations

# Executable "hello" depends on the files hello.o and run.o.

hello: hello.o start.o

	gcc -m32 -g -Wall -o hello hello.o start.o

 # Depends on the source and header files

hello.o: HelloWorld.c HelloWorld.h

	gcc -g -Wall -m32 -c -o hello.o HelloWorld.c

start.o: start.s

	nasm -g -f elf -w+all -o start.o start.s

 #tell make that "clean" is not a file name!

.PHONY: clean

 #Clean the build directory

clean:

	rm -f *.o hello
```

Important - the space you see to the left of some lines are tabs, not space characters.

Makefiles are just simple text files. You can create them with any text editor (notepad, kwrite, gedit,...). Note that some editors replace tabs with spaces, and you should beware of that.