UNX511/DPS912 – Makefile Creation

Simple Makefile Creation

Let us create a math executable based on four files:
Math.cpp, Conversions.cpp General.cpp and Geometry.cpp

To compile these, we would execute the following on the command line:

```
$ g++ Math.cpp General.cpp Conversions.cpp Geometry.cpp -o Math
```

Let us create a Makefile to do this for us. First of all, let us define a few variables: one for the compiler and one for flags. We really do not need to do this for small Makefiles, but it is good practice for larger ones. For instance, if we change our compiler or our flags, we only have to change it in one place.

```
CC=g++
CFLAGS=-I
CFLAGS+=-Wall
```

-I represents an include path, and **-Wall** means enable all warnings.

Let us define a variable that represents a task that performs the compilation. This task is called a rule:

```
Math: Math.cpp Conversions.cpp General.cpp Geometry.cpp
	$(CC) $(CFLAGS) Math.cpp Conversions.cpp General.cpp Geometry.cpp -o Math
```

After **Math:** is a list of C/C++ files. This means create a build if one of these files changes. If none of these files change, then the compiler will report:
make: Nothing to be done for 'Math'.

Note that the second line of the rule Maths: looks just like the command line version of the compilation. Also, note that the line after **Math:** starts with a **TAB**. **Spaces are not acceptable!**
Let us now define a rule that cleans up all object and executable files in the given directory:

```
clean:
	rm -f *.o Math
```

And let us define a rule that compiles all targets. In our case there is only one target:

```
all: Math
```

All together now we have a Makefile with the following syntax:

```
CC=g++
CFLAGS=-I
CFLAGS+=-Wall

Math: Math.cpp Conversions.cpp General.cpp Geometry.cpp
	$(CC) $(CFLAGS) Math.cpp Conversions.cpp General.cpp Geometry.cpp -o Math

clean:
	rm -f *.o Math

all: Math
```

UNX511/DPS912 – Makefile Creation

Note that there are three rules in this Makefile: **Math**, **clean** and **all**.

We could create another variable that holds a list of files to compile:

```
FILES=Math.cpp
FILES+=Conversions.cpp
FILES+=General.cpp
FILES+=Geometry.cpp
```

In which case our Makefile now becomes:

```
CC=g++
CFLAGS=-I
CFLAGS+=-Wall
FILES=Math.cpp
FILES+=Conversions.cpp
FILES+=General.cpp
FILES+=Geometry.cpp

Math: $(FILES)
	$(CC) $(CFLAGS) $(FILES) -o Math

clean:
	rm -f *.o Math

all: Math
```

Our compiler knows to run the Makefile script whenever make is invoked from the command line. For instance, make all will invoke the rule **all**. This is what you will see on the command line:

```
$ make all
g++ -I -Wall Math.cpp Conversions.cpp General.cpp Geometry.cpp -o Math
$
```

The executable Math has been created. If you invoke make clean, you will see:

```
$ make clean
rm -f *.o Math
$
```

And make clean all will produce the following:

```
$ make clean all
rm -f *.o Math
g++ -I -Wall Math.cpp Conversions.cpp General.cpp Geometry.cpp -o Math
$
```

If we invoke ls to see the list of files in our directory, we will see:

```
$ ls
Conversions.cpp  General.cpp  Geometry.cpp  Makefile  Math.cpp
Conversions.h    General.h    Geometry.h    Math      Math.h
$
```

UNX511/DPS912 – Makefile Creation

Makefile to Create a Static Library

Let us create a static library from our components that excludes our main() function in Math.cpp but includes those functions that appear in Conversions.cpp, General.cpp, and Geometry.cpp. So, we need to build a library that contains the object files from Conversions.cpp, General.cpp and Geometry.cpp.

Let us use the same variables as before: one for the compiler, one for the flags, and one for the list of C/CPP files:

```
CC=g++
CFLAGS=-I
CFLAGS+=-Wall
CFLAGS+=-c
FILES+=Conversions.cpp
FILES+=General.cpp
FILES+=Geometry.cpp
```

Note the addition to **CFLAGS**. The **-c** switch means create object files only.
Since we are creating a library from object files, let's create another variable to represent these object files:

```
OBJFILES+=Conversions.o
OBJFILES+=General.o
OBJFILES+=Geometry.o
```

And finally, to create the static library we are going to use an archiver utility called ar with the flags rcs:

```
AR=ar
ARFLAGS=rcs
```

The rule for creating the object file becomes:

```
Math: $(FILES)
	$(CC) $(CFLAGS) $(FILES)
```

And the rule for creating the static library becomes:

```
lib: $(OBJFILES)
	$(AR) $(ARFLAGS) libMath.a $(OBJFILES)
```

The rule for clean becomes:

```
clean:
	rm -f *.o *.a
```

And the rule for all becomes:

```
all: Math lib
```

UNX511/DPS912 – Makefile Creation

Putting it all together, the Makefile to create a static library with objects from Conversions.cpp, General.cpp and Geometry.cpp is:

```
CC=g++
CFLAGS=-I
CFLAGS+=-Wall
CFLAGS+=-c
FILES+=Conversions.cpp
FILES+=General.cpp
FILES+=Geometry.cpp
OBJFILES+=Conversions.o
OBJFILES+=General.o
OBJFILES+=Geometry.o
AR=ar
ARFLAGS=rcs

Math: $(FILES)
	$(CC) $(CFLAGS) $(FILES)

lib: $(OBJFILES)
	$(AR) $(ARFLAGS) LibMath.a $(OBJFILES)

clean:
	rm -f *.o *.a

all: Math lib
```

Note that there are now four rules in our Makefile: **Math**, **lib**, **clean**, and **all**.
If we invoke make clean all, we will see:

```
$ make clean all
rm -f *.o *.a
g++ -I -Wall -c Conversions.cpp General.cpp Geometry.cpp
ar rcs LibMath.a Conversions.o General.o Geometry.o
$
```

If we invoke ls to see the list of files in our directory, we will see:

```
$ ls
Conversions.cpp  Conversions.o  General.h  Geometry.cpp  Geometry.o  Math.h
Conversions.h    General.cpp    General.o  Geometry.h    Makefile    LibMath.a
$
```

We are not done. Let us create a rule to install this library. This means, copying the static library LibMath.a and its header files to a common place where it can be used by other programs.

```
install:
	cp LibMath.a /home/miguelwatler/src/Week2/Maths
	cp *.h /home/miguelwatler/src/Week2/Maths
```

Note that I got lazy and used *.h to copy all header files in this directory. It would be better to create a list of header files that I want to copy.

UNX511/DPS912 – Makefile Creation

The Makefile now becomes:

```
CC=g++
CFLAGS=-I
CFLAGS+=-Wall
CFLAGS+=-c
FILES+=Conversions.cpp
FILES+=General.cpp
FILES+=Geometry.cpp
OBJFILES+=Conversions.o
OBJFILES+=General.o
OBJFILES+=Geometry.o
AR=ar
ARFLAGS=rcs

Math: $(FILES)
	$(CC) $(CFLAGS) $(FILES)

lib: $(OBJFILES)
	$(AR) $(ARFLAGS) LibMath.a $(OBJFILES)

install:
	cp LibMath.a /home/miguelwatler/src/Week2/Maths
	cp *.h /home/miguelwatler/src/Week2/Maths

clean:
	rm -f *.o *.a

all: Math lib
```

A make clean all install from the command line results in:

```
$ make clean all install
rm -f *.o *.a
g++ -I -Wall -c Conversions.cpp General.cpp Geometry.cpp
ar rcs LibMath.a Conversions.o General.o Geometry.o
cp LibMath.a /home/miguelwatler/src/Week2/Maths
cp *.h /home/miguelwatler/src/Week2/Maths
$
```

An ls of the install directory gives the static library and all its header files. Therefore to use the Maths library that we created, all we need to provide is the static library and header files:

```
miguelwatler@ubuntu:~/src/Week2/Maths$ ls
Conversions.h  General.h  Geometry.h  Math.h  LibMath.a
miguelwatler@ubuntu:~/src/Week2/Maths$
```

UNX511/DPS912 – Makefile Creation

You can execute the command nm on LibMath.a to see the binaries inside:

```
miguelwatler@ubuntu:~/src/Week2/Maths$ nm libMath.a

Conversions.o:
00000000000000a0 T _Z17KilogramsToPoundsd
00000000000000bc T _Z17PoundsToKilogramsd
0000000000000000 T _Z19CelsiusToFahrenheitd
0000000000000084 T _Z19CentimetersToInchesd
0000000000000034 T _Z19FahrenheitToCelsiusd
0000000000000068 T _Z19InchesToCentimetersd

General.o:
0000000000000032 T _Z5powerdi
0000000000000076 T _Z8absoluted
0000000000000000 T _Z9factoriali

Geometry.o:
0000000000000000 T _Z13Circumferenced
000000000000001c T _Z4Aread
000000000000003d T _Z6Volumed
miguelwatler@ubuntu:~/src/Week2/Maths$
```

This means that the following functions are DEFINED inside of LibMath.a:

KilogramsToPounds
PoundsToKilograms
CelsiusToFahrenheit
CentimetersToInches
FahrenheitToCelsius
InchesToCentimeters
power
absolute
factorial
Circumference
Area
Volume

The symbols or characters that you see after the function name refer to the type(s) of the parameters.

UNX511/DPS912 – Makefile Creation

Using the Static Library

Let us create a small application to use the static library. Let the application test every function in this library. These functions are declared in the header files. If I do a **cat** on each header file I see:

```
miguelwatler@ubuntu:~/src/Week2/Maths$ cat Conversions.h
//Conversions.h - Header file for Simple Unit Conversions
//
// 14-Jan-19  M. Watler        Created.

double CelsiusToFahrenheit(double celsius);
double FahrenheitToCelsius(double fahrenheit);
double InchesToCentimeters(double inches);
double CentimetersToInches(double centimeters);
double KilogramsToPounds(double kilograms);
double PoundsToKilograms(double pounds);

miguelwatler@ubuntu:~/src/Week2/Maths$ cat General.h
//General.h - Header file for General Calculations
//
// 14-Jan-19  M. Watler        Created.

int factorial(int num);
double power(double num, int exponent);
double absolute(double num);

miguelwatler@ubuntu:~/src/Week2/Maths$ cat Conversions.h
//Conversions.h - Header file for Simple Unit Conversions
//
// 14-Jan-19  M. Watler        Created.

double CelsiusToFahrenheit(double celsius);
double FahrenheitToCelsius(double fahrenheit);
double InchesToCentimeters(double inches);
double CentimetersToInches(double centimeters);
double KilogramsToPounds(double kilograms);
double PoundsToKilograms(double pounds);

miguelwatler@ubuntu:~/src/Week2/Maths$ cat Math.h
//Math.h - header file for the math library
//
// 14-Jan-19  M. Watler        Created.

#include "General.h"
#include "Geometry.h"
#include "Conversions.h"
miguelwatler@ubuntu:~/src/Week2/Maths$
```

UNX511/DPS912 – Makefile Creation

Let us call the test application MathTest.cpp and put it in a separate directory called MathTest. We need to create a list of C/CPP files. We now need to use the include path to tell our MathTest.cpp where the header files are, and we need to create a new variable specifying LibMath.a and the path to it:

```
CFLAGS=-I/home/miguelwatler/src/Week2/Maths
CFLAGS+=-Wall
FILES=MathTest.cpp
LIBS = -L/home/miguelwatler/src/Week2/Maths -lMath
```

Note that the library is denoted by **-lMath**. The switch **-l** assumes that the library name is prefixed by **"lib"**. Therefore **-lMath** really means **libMath.a**. We could have defined LIBS as follows:

```
LIBS =/home/miguelwatler/src/Week2/Maths/libMath.a
```

We have to create rules as before. To build MathTest we need to compile our C/CPP files and link it/them with the static library:

```
MathTest: $(FILES)
	$(CC) $(CFLAGS) $(FILES) -o MathTest $(LIBS)
```

We need to create a rule for clean and also one for all, as before. Our entire Makefile is therefore:

```
CC=g++
CFLAGS=-I/home/miguelwatler/src/Week2/Maths
CFLAGS+=-Wall
FILES=MathTest.cpp
LIBS = -L/home/miguelwatler/src/Week2/Maths -lMath

MathTest: $(FILES)
	$(CC) $(CFLAGS) $(FILES) -o MathTest $(LIBS)

clean:
	rm -f *.o MathTest

all: MathTest
```

A make clean all will look like:

```
miguelwatler@ubuntu:~/src/Week2/MathTest$ make clean all
rm -f *.o MathTest
g++ -I/home/miguelwatler/src/Week2/Maths -Wall MathTest.cpp -o MathTest -
L/home/miguelwatler/src/Week2/Maths -lMath
miguelwatler@ubuntu:~/src/Week2/MathTest$
```

An ls on the current directory will provide:

```
miguelwatler@ubuntu:~/src/Week2/MathTest$ ls
Makefile  MathTest  MathTest.cpp
miguelwatler@ubuntu:~/src/Week2/MathTest$
```

UNX511/DPS912 – Makefile Creation

Creating a Shared Library

Creating a shared library is similar to creating a static library. One can execute two g++/gcc commands to accomplish this. The first compiles the C/CPP files with the option fPIC which means compile the code with position independent code (pic, or PIC) which is required for a shared library:

```
$ g++ -I -fPIC -Wall -c Conversions.cpp General.cpp Geometry.cpp
```

The second creates the shared library with the option -shared:

```
$ g++ -shared -o libMath.so Conversions.o General.o Geometry.o
```

Shared libraries have to be known by the system. They should be put under **/usr/lib** with the other libraries (to do so you will need root permissions). If they are put somewhere else, the path has to be exported and the command ldconfig executed to inform the system of its location. For shared libraries, typically the shared library and header files are in different directories completely.

The Makefile to create a shared library from Conversions.cpp, General.cpp and Geometry.cpp therefore looks like:

```
CC=g++
CFLAGS=-I
CFLAGS+=-fPIC
CFLAGS+=-Wall
CFLAGS+=-c
LIBFLAGS=-shared
LIBFLAGS+=-o
FILES=Conversions.cpp
FILES+=General.cpp
FILES+=Geometry.cpp
OBJFILES=Conversions.o
OBJFILES+=General.o
OBJFILES+=Geometry.o

Math: $(FILES)
	$(CC) $(CFLAGS) $(FILES)

lib: $(OBJFILES)
	$(CC) $(LIBFLAGS) libMath.so $(OBJFILES)

install:
	cp libMath.so /usr/lib
	cp *.h /home/miguelwatler/src/Week2/Maths/shared

clean:
	rm -f *.o *.so

all: Math lib
```

UNX511/DPS912 – Makefile Creation

A make clean all install will look like:

```
root@ubuntu:/home/miguelwatler/src/Week2/MathLib/shared# make clean all install
rm -f *.o *.so
g++ -I -fPIC -Wall -c Conversions.cpp General.cpp Geometry.cpp
g++ -shared -o libMath.so Conversions.o General.o Geometry.o
cp libMath.so /usr/lib
cp *.h /home/miguelwatler/src/Week2/Maths/shared
root@ubuntu:/home/miguelwatler/src/Week2/MathLib/shared#
```

Using a Shared Library

When compiling with a shared library, the syntax is exactly the same as with a static library. we need to create a variable specifying LibMath.so and the path to it:

```
LIBS = -L/usr/lib -lMath
```

Our Makefile therefore looks like:

```
CC=g++
CFLAGS=-I/home/miguelwatler/src/Week2/Maths/shared
CFLAGS+=-Wall
FILES=MathTest.cpp
LIBS = -L/usr/lib -lMath

MathTest: $(FILES)
	$(CC) $(CFLAGS) $(FILES) -o MathTest $(LIBS)

clean:
	rm -f *.o MathTest

all: MathTest
```

A make clean all will look like:

```
miguelwatler@ubuntu:~/src/Week2/MathTest/shared$ make clean all
rm -f *.o MathTest
g++ -I/home/miguelwatler/src/Week2/Maths/shared -Wall MathTest.cpp -o MathTest -L/usr/lib -lMath
miguelwatler@ubuntu:~/src/Week2/MathTest/shared$
```

To display shared object depencencies on MathTest (see libMath.so below), execute the ldd command:

```
miguelwatler@ubuntu:~/src/Week2/MathTest/shared$ ldd MathTest
	linux-vdso.so.1 (0x00007fffe69fb000)
	libMath.so => /usr/lib/libMath.so (0x00007f1e61340000)
	libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007f1e60fb0000)
	libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f1e60bb8000)
	libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f1e60818000)
	/lib64/ld-linux-x86-64.so.2 (0x00007f1e61750000)
	libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f1e60600000)
miguelwatler@ubuntu:~/src/Week2/MathTest/shared$
```

UNX511/DPS912 – Makefile Creation

More Information

For more information on Makefiles see:
ftp://ftp.gnu.org/old-gnu/Manuals/make-3.79.1/html_chapter/make_2.html

For more information on static and shared libraries see:
https://renenyffenegger.ch/notes/development/languages/C-C-plus-plus/GCC/create-libraries/index
https://www.thegeekstuff.com/2012/06/linux-shared-libraries/