Part 1: Fixing code indentation problems

If you are writing your code using `gedit`, you could end up with a badly indented code pretty quickly! There is a utility program called `indent` that you can use to format your code. The indent tool has many customizable styles of code formatting that can be tweaked as you need. I personally like to use indent with `-kr -cli4 -l80 -nut -bl -bli0 -nce` options. I have created a one-line bash script with the aforementioned style options that you can use to format your code.

WARNING: Be careful when you use this command line tool,

- Save your original file in the text editor before calling the `indent.sh` script.
- To be extra cautious, make a backup of your program the first time, so that you don't loose your work if something goes wrong.
- Reload the document in your text editor after you run the script to see the changes.

To run the script, open a terminal window, navigate to the location of your program using `cd` command and then issue the following command:

```
$ ~/cse340/tools/indent.sh file.c
```

You should change `file.c` with the name of the file you want to re-indent.

Part 2: Generating simple Makefile

To generate a `Makefile` for your C/C++ project, you can use the Python script named `generate_simple_makefile.py`.

For example, let's imagine that we have a C++ project with the following files:

```
example/
|--- lexer.h
|--- lexer.cc
|--- main.cc
\--- parser.cc
```

We can generate a Makefile for our project by using the following command in a terminal:

```
$ cd /path/to/example
$ ~/cse340/tools/generate_simple_makefile.py *.h *.cc > Makefile
```

The generated Makefile should look like this:

```
# Automatically generated on Mon, May 16 2016 18:15:35

DEP = .deps
CC  = gcc
CXX = g++
CFLAGS   = -Wall
CXXFLAGS = -Wall

# Target
SOURCES  = lexer.cpp main.cpp parser.cpp
OBJECTS  = lexer.o main.o parser.o
DEPFILES = $(patsubst %.o,$(DEP)/%.d,$(OBJECTS))
TARGET   = a.out
ZIP_FILE = source.zip

.PHONY: all clean zip

all: $(DEP) $(TARGET)

-include $(DEPFILES)
```

```
$(TARGET): $(OBJECTS)
	@echo "Linking $@"
	$(CXX) $(OBJECTS) -o $@ $(LIBS)

%.o: %.c
	@echo "Compiling $*.c"
	$(CC) -c $(CFLAGS) $*.c $(INCLUDE) -o $@
	@$(CC) -MM -MP -MT $@ $(CFLAGS) $*.c $(INCLUDE) > $(DEP)/$*.d

%.o: %.cpp
	@echo "Compiling $*.cpp"
	$(CXX) -c $(CXXFLAGS) $*.cpp $(INCLUDE) -o $@
	@$(CXX) -MM -MP -MT $@ $(CXXFLAGS) $*.cpp $(INCLUDE) >
$(DEP)/$*.d

$(DEP):
	@mkdir -p $@

zip: $(SOURCES)
	@echo "Zipping source files to $(ZIP_FILE)"
	@zip $(ZIP_FILE) Makefile $(SOURCES) lexer.h

clean:
	@rm -f $(TARGET) $(DEPFILES) $(OBJECTS) $(ZIP_FILE)
	@rmdir $(DEP)
```

You can use the generated Makefile to compile your program and also to make a zip archive of your code. To compile your program use the default target of the Makefile:

```
$ make
```

That above command will generate something like the following for our example project:

```
  Compiling lexer.cpp
  g++ -c -Wall  lexer.cpp  -o lexer.o
  Compiling main.cpp
  g++ -c -Wall  main.cpp  -o main.o
  Compiling parser.cpp
  g++ -c -Wall  parser.cpp  -o parser.o
  Linking a.out
  g++ lexer.o main.o parser.o -o a.out
```

You can also prepare a zip file containing all your source code files with the following command:

```
$ make zip
```

For our example project, the output is:

```
Zipping source files to source.zip
  adding: Makefile (deflated 50%)
  adding: lexer.cpp (stored 0%)
  adding: main.cpp (stored 0%)
  adding: parser.cpp (stored 0%)
  adding: lexer.h (stored 0%)
```

To cleanup the project directory and remove all files created by make and make zip, you can use the following command:

```
$ make clean
```

The `generate_simple_makefile.py` script has many options that can be used to customize the generated Makefile to a certain degree. You can get a list of these options by invoking the help command:

```
$ ~/cse340/tools/generate_simple_makefile.py -h
```