Isrish's Blog

Just version of e-me!

makefile for OpenCV C++ project

Filed under: Uncategorized — Leave a comment
October 17, 2012

One of the features of C and C++ that's considered a strength is the idea of "separate compilation". Instead of writing all the code in one file, and compiling that one file, C/C++ allows you to write many .cpp files and compile them separately.

A good programing flow is to have a modular structure having a **.cpp** files and a corresponding **.h** file. A **.cpp** usually consists of:

- the implementations of all methods in a class,
- standalone functions (functions that aren't part of any class),
- and global variables (usually avoided).

The corresponding **.h** file contains

- class declarations,
- function prototypes,
- and extern variables (again, for global variables).

The purpose of the **.h** files is to export "services" to other **.cpp** files.

Why all the talk about how **.cpp** files get compiled in C++? Basicaly in linux you will use gcc or g++ on the command line or if you are using IDE it will take care for you based on your project setting.

For example :The trivial way to compile the files and obtain an executable, is by running the command like

```
1 | g++ main.cpp hello.cpp factorial.cpp -o hello
```

Compiling your source code files can be tedious, specially when you want to include several source files and have to type the compiling command every time you want to do it. Makefiles are a simple way to organize code compilation. This blog post does not even scratch the surface of what is possible using *make*, but is intended as a starters guide so that you can quickly and easily create your own makefiles for small to medium-sized projects. Because of the way C++ compiles files, makefiles can take advantage of the fact that when you have many **.cpp** files, it's not necessary to recompile all the files when you make changes. You only need to recompile a small subset of the file. Although it's much faster to compile now, it's still not very fast. where recompiling the entire code can take many hours, you will still want a

makefile to avoid having to recompile everything.

A makefile is based on a very simple concept. A makefile typically consists of many entries. Each entry has:

- a target (usually a file and can have many targets)
- the dependencies (files which the target depends on)
- and commands to run, based on the target and dependencies.

In this post I will show how to write make file for OpenCV based c++ project which have two target file , lets say one webcam and the other dataset.

1. Compiler to use , in this case g++

```
1 | CXX=g++
```

2. Options I'll pass to the compiler

```
1 | CXXFLAGS = -I/usr/local/include/opencv2  -g3 -Wall -c
```

-I/usr/local/include/opencv2 Here the include dir for opencv2.4.x library is passed as an option. It is found at /usr/local/include/opencv2 Put the correct include dir if you have a different one

-Wall option is very nice it will show all the warning, that may result in possible bug in the program
-c option Compile or assemble the source files, but do not link. The linking stage simply is not done. The ultimate output is in the form of an object file for each source file
-g3 Request debugging information and also use level to specify how much information. The default level is 2. Here Level 3 includes extra information, such as all the macro definitions present in the program.

Linking libraries files need to compile opencv based application. At least for a small application we need the following basics one as listed, add more for example to add the legacy support write -lopencvlegacy_support.

```
1 | LIBS = -L/usr/lib -lpthread -ldl -lm -std=gnu++0x -std=c++0x -lopencv_cc
```

4. Put the soruce files Here as an example face.cpp and function.cpp are commom source files used by both targets(webcam and datasettest)

```
1 | COMMON_SOURCES = face.cpp functions.cpp
2 |  WEBCAM_SOURCES = webcam.cpp processWebcam.cpp
3 |  DATASET_SOURCES = datasettest.cpp
```

5. Object files

```
1 | COMMON_OBJECTS = face.o functions.o
2 | WEBCAM_OBJECTS = webcam.o processWebcam.o
3 | DATASET_OBJECTS = datasettest.o
```

6. Our two target excutable

```
1 | WEB_EXECUTABLE = webcamEC
2 | DATASET_EXECUTABLE = datasetEC
```

7. The real mess. Read more on the Internet about makefile to scratch details about them, for a hint I have putted a comment lines

```
.PHONY: all webcam dataset
# for make all
all: webcam dataset single
#for making webcam
webcam: $(WEB_EXECUTABLE)
# for makinf dataset
dataset: $(DATASET_EXECUTABLE)

$(WEB_EXECUTABLE): $(COMMON_OBJECTS) $(WEBCAM_OBJECTS)
@echo 'Building WEB_EXECUTABLE='
$(CXX) -o $(WEB_EXECUTABLE) $(COMMON_OBJECTS) $(WEBCAM_OBJECTS) $(LIBS)
@echo 'Building Finished:'

$(DATASET_EXECUTABLE): $(COMMON_OBJECTS) $(DATASET_OBJECTS)
@echo 'Building DATASET_EXECUTABLE='
$(CXX) -o $(DATASET_EXECUTABLE) $(COMMON_OBJECTS) $(DATASET_OBJECTS) $(
@echo 'Building Finished:'
# for make clean , cleaning all the build forceuly delete all the files
clean:
rm -rf $(WEB_EXECUTABLE) $(DATASET_EXECUTABLE) $(COMMON_OBJECTS) $(WEBC
@echo 'Clean FInished!'

%.d: %.cpp
$(CXX) -MM -MF $@ $<

%.o: %.d

-include $(OBJS:.o=.d)
```

8 . Finally putting all things together we have a make file that can build to executable.

```
# 1.
Cxx=g++
# 2.
CXXFLAGS = -I/usr/local/include/opencv2 -O0 -g3 -Wall -c
# 3.
LIBS =    -L/usr/lib -lpthread -ldl -lm -std=gnu++0x -std=c++0x -lopenc

# 4.
# put the soruce files here example here face.cpp and function.cpp are
COMMON_SOURCES = face.cpp functions.cpp
WEBCAM_SOURCES = webcam.cpp processWebcam.cpp
DATASET_SOURCES = datasettest.cpp

# 5.Object files
COMMON_OBJECTS = face.o functionSets.o
WEBCAM_OBJECTS = webcam.o processWebcam.o
DATASET_OBJECTS = datasettest.o

#6. two target excutable
WEB_EXECUTABLE = webcamEC
DATASET_EXECUTABLE = datasetEC

# 7. the real mess
.PHONY: all webcam dataset
# for make all
all: webcam dataset single
#for making webcam
webcam: $(WEB_EXECUTABLE)
# for makinf dataset
dataset: $(DATASET_EXECUTABLE)

$(WEB_EXECUTABLE): $(COMMON_OBJECTS) $(WEBCAM_OBJECTS)
@echo 'Building WEB_EXECUTABLE='
$(CXX) -o $(WEB_EXECUTABLE) $(COMMON_OBJECTS) $(WEBCAM_OBJECTS) $(LIBS)
@echo 'Building Finished:'

$(DATASET_EXECUTABLE): $(COMMON_OBJECTS) $(DATASET_OBJECTS)
@echo 'Building DATASET_EXECUTABLE='
$(CXX) -o $(DATASET_EXECUTABLE) $(COMMON_OBJECTS) $(DATASET_OBJECTS) $(
@echo 'Building Finished:'
# for make clean , cleaning all the build forceuly delete all the files
clean:
rm -rf $(WEB_EXECUTABLE) $(SINGLEFILE_EXECUTABLE) $(DATASET_EXECUTABLE)
@echo 'Clean FInished!'

%.d: %.cpp
$(CXX) -MM -MF $@ $<

%.o: %.d

-include $(OBJS:.o=.d)
```

You can use command line

```
root@root:~$ make all
root@root:~$ make webcamEC
root@root:~$ make datasetEC
root@root:~$ make clean
```

This is just one way of having makefile, there is alot of magice makefile can do.For now as a basic the above is more than enough.