Tutorial of GCC , Make and CMake.

GCC, Make and CMake are famous and useful tools to build C/C++ programs.

This note helps you to understand functions and how to use them properly.

Sample program

Task : Calculate PI in 3 different methods.

1. Viete's method
2. Ramanujan's method
3. Gregory's method

Directory tree

- cmake_tutorial
  - include
    - factorial.hpp
    - gregory.hpp
    - ramanujan.hpp
    - viete.hpp
  - src
    - main.cpp
    - factorial.cpp
    - gregory.cpp
    - ramanujan.cpp
    - viete.cpp

Now, we build the above program in three different ways.

1. gcc
2. gcc + make
3. gcc + make + cmake

Way 1. Build with gcc

What is GCC?

It is a famous compiler of C, C++ and some other programming languages.

Environment setup

Install gcc.

- Windows :
  - `scoop install gcc`
- Mac OS :

    - `brew install gcc`
- Ubuntu :
    - `sudo apt install build-essential`

Procedures

1. Move to the directory to output intermediate files and executables.
    - `cd build`
2. Build shared libraries
    - `g++ -shared -o libV.dll -I../include -c ../src/viete.cpp`
    - `g++ -shared -o libR.dll -I../include -c ../src/ramanujan.cpp`
    - `g++ -shared -o libG.dll -I../include -c ../src/gregory.cpp`
    - Use ".so" extention instead when your OS is linux or mac.
3. Build a static library.
    - `g++ -static -o factorial.o -c -I../include ../src/factorial.cpp`
    - `ar rcs libfactorial.a factorial.o`
4. Generate executable.
    - `g++ -o calcpi ../src/main.cpp -I../include -L. -lV -lR -lG -lfactorial`
    - Note : -I option indicates the location of header files.
    - Note : -L option indicates the location of shared libraries.
5. Run `./calcpi`

Way 2. Build with gcc + make

What is Make?

It is an useful tool to help compiling C/C+ programs.

Advantages of make

- Fewer build commands. Just type "make".
- Avoid unnecessary build processes => Faster builds!

Environment setup

Install gcc and make.

- Windows :
    - `scoop install gcc make`
- Mac OS :
    - `brew install gcc make`
- Ubuntu :
    - `sudo apt install build-essential`

Procedures

1. `cd build` and locate Makefile

```
# compiler
CC  = g++
# compile options
CFLAGS    = -Wall # enable debugger
# name of executable
TARGET  = calcpi
# target src code
SRCS    = ../src/main.cpp
# src directory
SRCDIR  = ../src
# include directory
INCDIR  = -I../include
# directory including libraries
LIBDIR  = -L.
# library files to link
LIBS    = -lV -lR -lG -lfactorial

# Generate an executable.
$(TARGET): $(SRCS) libV.so libR.so libG.so libfactorial.a
    $(CC) $(CFLAGS) -o $@ $(SRCS) $(INCDIR) $(LIBDIR) $(LIBS)

# Build libraries
libV.so :
    $(CC) -shared -o $@ $(INCDIR) -c $(SRCDIR)/viete.cpp

libR.so :
    $(CC) -shared -o $@ $(INCDIR) -c $(SRCDIR)/ramanujan.cpp

libG.so :
    $(CC) -shared -o $@ $(INCDIR) -c $(SRCDIR)/gregory.cpp

libfactorial.a :
    $(CC) -static -o factorial.o $(INCDIR) -c $(SRCDIR)/factorial.cpp
    ar rcs $@ factorial.o

# make all
all: clean $(TARGET)

# make clean
clean:
    -rm -f $(TARGET) *.dll *.so *.o
```

2. `make`
3. Run `./calcpi`

Way 3. Build with gcc + make + cmake

What is CMake?

It is an excellent tool extending make to help building C/C++ projects.

Advantages of CMake

- Easier to set up build environment for complexed projects of C/C++.
  - CMake generates Makefile automatically.

For example,

- Better support for complex directory structures.
- Easy to specify dependant libraries.
- Multiple executables can be generated at once.

Environment setup

Install gcc, make, and cmake.

- Windows :
  - `scoop install gcc make cmake`
- Mac OS :
  - `brew install gcc make cmake`
- Ubuntu :
  - `sudo apt install build-essential cmake`

Procedures

1. `cd cmake_tutorial` and locate CMakeLists.txt

```
# set required version
cmake_minimum_required(VERSION 3.1)
# set compiler
set(CMAKE_C_COMPILER gcc)
set(CMAKE_CXX_COMPILER g++)
# set project name
project( CALCPI CXX)

# set build options
# set(CMAKE_CXX_FLAGS "-g")# Debug mode
set(CMAKE_CXX_FLAGS "-O2 -march=native -std=c++11 -Wall")# release
mode

# set include directories
include_directories(${CMAKE_CURRENT_SOURCE_DIR}/include)
include_directories(${CMAKE_CURRENT_SOURCE_DIR}/src)

# generate shared libraries
add_library(viete     SHARED ${CMAKE_CURRENT_SOURCE_DIR}/src/viete.cpp
)
add_library(ramanujan SHARED
```

```
${CMAKE_CURRENT_SOURCE_DIR}/src/ramanujan.cpp)
add_library(gregory   SHARED
${CMAKE_CURRENT_SOURCE_DIR}/src/gregory.cpp  )

# generate a static library
add_library(factorial STATIC
${CMAKE_CURRENT_SOURCE_DIR}/src/factorial.cpp)

# generate executables
add_executable(calcpi ${CMAKE_CURRENT_SOURCE_DIR}/src/main.cpp)

# link libraries
target_link_libraries(calcpi viete ramanujan gregory factorial)
```

2. `cd build`
3. `cmake ..`
4. `make`
5. Run `./calcpi`