1) CMake 사용법, 2) Ceres-Solver 설치, 3) HelloWorld 예제

RAIL 학부연구생
20이재원

1) CMake 사용법

컴파일 : 소스코드 -> 바이너리(번역) -> 실행파일

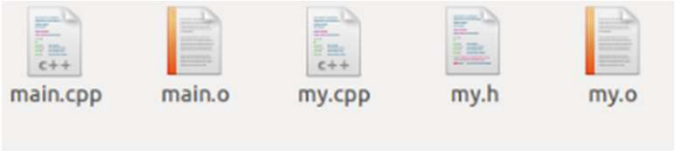
gcc : 모든 리눅스에서 공통으로 쓰는 컴파일러 모음

1) 기본적인 빌드 과정

`g++ -c main.cpp` -> 소스 코드 파일을 통해 오브젝트 파일 생성, 곧바로 실행할 수 없다.

`g++ -c my.cpp`

`g++ -o test main.o my.o` -> 오브젝트 파일을 통해 실행파일(test) 생성.
합치고자 하는 파일들을 모두 적어줘야 한다.



-> 매번 모든 파일을 일일이 해주기 힘들다.

-> Makefile

Reference : https://ladofa.blogspot.com/2018/07/c-1.html

1) CMake 사용법 - Makefile

<Makefile> 형식

[타깃이름]: [타깃에 필요한 파일들]
[타깃 실행 코드]

```
my.o : my.h my.cpp
        g++ -c my.cpp
```

-> `$ make my.o`

-- 타깃을 make로 실행시키면 makefile에 타깃 실행 코드를 수행한다.

```
my.o: my.h my.cpp
    g++ -c my.cpp

main.o: my.h main.cpp
    g++ -c main.cpp

test: my.o main.o
    g++ -o test main.o my.o

all: test

clear:
    rm -f my.o main.o test
```

-> all은 타깃 실행 코드가 없기에 아무 동작도 수행하지 않는다.

-> clear는 타깃 파일이 없다. -> 무조건 명령 수행

Reference : https://ladofa.blogspot.com/2020/08/c-2.html

1) CMake 사용법 – CMake, CMakeLists.txt

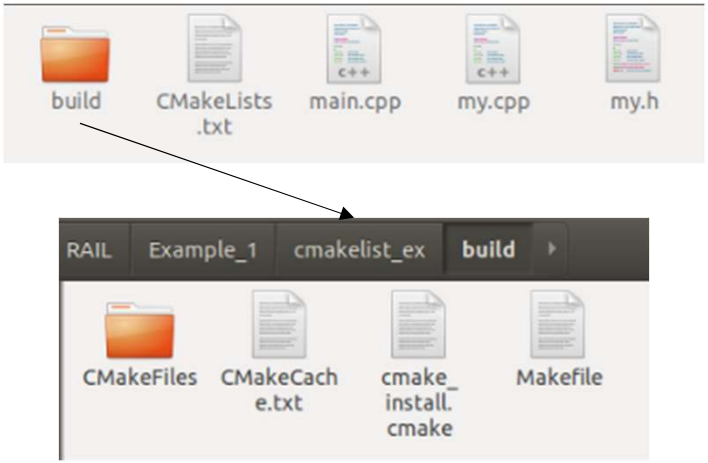
Makefile -> 파일양이 많아지고 의존성이 방대해지면 일일이 gcc명령을 내리기 힘들다

Cmake는 Makefile과 같은 작업들을 단순화하여 간단한 명령어를 통해 build할 수 있도록 한다.

<CMakeLists.txt> 형식

```
cmake_minimum_required (VERSION 3.10)
project (mytest)
add_executable (mytest my.cpp main.cpp)
```

<CMakeLists.txt> 구조



Build파일은 파일개수가 많아져 구분하기 위해 만든 파일일뿐이다.

$ cmake .. 을 통해 build안에 Makefile을 생성한다. (..은 CMakeLists.txt의 경로)

$ make 를 통해 생성된 Makefile을 실행한다.

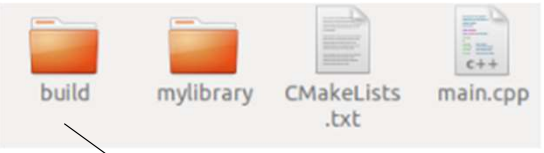
빌드에 성공하면 mytest실행파일이 생성된다.

소스코드를 수정하면 이제 $ make 만을 통해서 다시 빌드해준후 실행파일을 실행시킨다.

Reference : https://ladofa.blogspot.com/2020/09/c-3-cmake.html

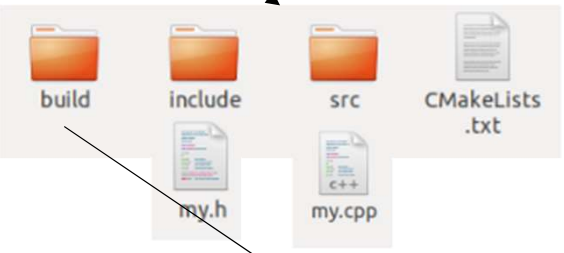
1) CMake 사용법 – CMake, CMakeLists.txt

CMakeLists.txt 구조화



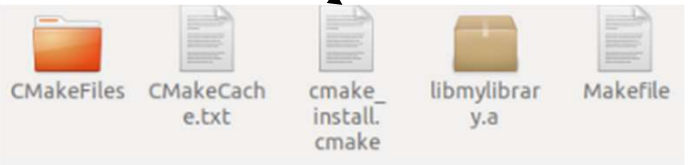
-- Library를 가지고 main.cpp의 Makefile을 생성할 수 있는 CMakeLists.txt

```
cmake_minimum_required (VERSION 3.10)
project (mytest)
add_subdirectory(mylibrary)
add_executable (mytest main.cpp) #실행Makefile 생성
target_link_libraries(mytest PUBLIC mylibrary)
```



-- my.h와 my.cpp를 라이브러리로 쓸 수 있는 CMakeLists.txt

```
cmake_minimum_required (VERSION 3.10)
project (mylibrary)
add_library (mylibrary src/my.cpp) #동적 라이브러리 생성
target_include_directories(mylibrary PUBLIC include)
```



-- 동적 라이브러리 생성됨.

공통적으로 build파일 생성하고 build파일에서 $ cmake .. 후 $ make

Reference : https://ladofa.blogspot.com/2020/09/c-3-cmake.html

2) Ceres-Solver 개요 및 설치 – What is Ceres Solver?

-- 오픈 소스 C++ 라이브러리이며 모델링 혹은 크고 복잡한 최적화 문제를 해결하는데 사용된다.

또한, Non-linear Least Squares problems를 해결하기 위해 사용된다.

-- Non-linear Least Squares
먼저 대부분의 문제는 비선형적인 형태를 해결해야한다. 선형적인 형태로도 데이터를 측정하고 데이터를 대표하는 함수를 만들 수 있지만 그 결과도 선형적이기에 데이터들을 대표하기에는 오차가 크다.

Least-Squares는 데이터를 대표하는 함수와 실제 여러 데이터들과의 오차의 제곱이 최소가 되게 만들겠다는 것이다.

즉, Non-linear Least Squares는 말 그대로 비선형적인 함수에 대해 오차가 최소가 되게 하는 테크닉이다.

개념

-- Non-linear Least Squares는 n개의 알려지지 않은 매개변수에서 m 관측 집합을 fit하는데 사용되는 방법.

-- 이 방법은 Nonlinear Regression의 일부 형태에서도 사용

-- 이 방법의 기반은 모델을 선형으로 근사하고 반복을 통해 매개변수를 세분화하는 것이다.

Reference : http://ceres-solver.org/, https://en.wikipedia.org/wiki/Non-linear_least_squares

2) Ceres-Solver 개요 및 설치 – Why?

- **Code Quality**
  4년이상 구글에서 개발에 사용됐으며, 명확하고 지속적으로 개발 및 관리되고 있다.
- **Modeling API**
- **Solver Choice**
  문제나 요구 사항에 따른 다양한 최적화 알고리즘이 제공됨.
- **Speed**
- **Solution Quality**
- **Estimation**
  민감도, 불확도를 평가 및 분석을 대규모로 진행할 수 있음.
- **Portability**
  Runs on Linux, Windows, Mac OS X, Android and iOS.
- **Community**

Reference : http://ceres-solver.org/features.html

2) Ceres-Solver 개요 및 설치 - Installation

In Terminal,

```
git clone https://ceres-solver.googlesource.com/ceres-solver

# Dependencies
# CMake
sudo apt-get install cmake
# google-glog + gflags
sudo apt-get install libgoogle-glog-dev libgflags-dev
# BLAS & LAPACK
sudo apt-get install libatlas-base-dev
# Eigen3
sudo apt-get install libeigen3-dev
# SuiteSparse and CXSparse (optional)
sudo apt-get install libsuitesparse-dev

mkdir ceres-bin
cd ceres-bin
cmake ../ceres-solver-2.0.0
make -j3
make test
# Optionally install Ceres, it can also be exported using CMake which
# allows Ceres to be used without requiring installation, see the documentation
# for the EXPORT_BUILD_DIR option for more information.
sudo make install
```

- Dependencies
- CMake 3.5 or later **required.**
- glog 0.3.1 or later. **Recommended**
- Eigen 3.3 or later **required.**
- Gflags
- SuiteSparse.**(Optional)**
- BLAS and LAPACK

Reference : http://ceres-solver.org/installation.html
Ubuntu 18.04.6 LTS, https://github.com/Lee-JaeWon/Ceres-Solver-Examples

3) HelloWorld Tutorial – Nonlinear Least Squares Introduction

(1)¶

$$\min_{\mathbf{x}} \quad \frac{1}{2}\sum_{i} \rho_i\left(\left\|f_i\left(x_{i_1}, \ldots, x_{i_k}\right)\right\|^2\right)$$
$$\text{s.t.} \quad l_j \leq x_j \leq u_j$$

Mean Squared Error (MSE)

$$cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left(H(x^{(i)}) - y^{(i)}\right)^2$$

Mean Prediction Target

다음과 같은 제한된 경계의 비선형적인 문제를 Ceres에서 해결할 수 있다.

$\left(\left\|f_i\left(x_{i_1}, \ldots, x_{i_k}\right)\right\|^2\right)$는 Cost function이며, ρ_i는 Loss function 이다.

- **Cost Function**
  모든 input data에 대해서 오차를 계산하는 함수

- **Loss Function**
  하나의 input data에 대해서 예측 값과 데이터 값의 오차를 계산하는 함수

→ Cost Function을 모든 데이터에 대해 계산한 Loss Function의 평균값으로 생각할 수 있다.

Reference : http://ceres-solver.org/nnls_tutorial.html#hello-world
https://lee-jaewon.github.io/deep_learning_study/Lec03(TensorFlow)/

3) HelloWorld Tutorial – Problem & Source code

Problem : $\frac{1}{2}(10-x)^2.$

First Step, 선형적인 작은 문제로 나누기 $f(x) = 10 - x$:

1. helloworld.cc

```
49  struct CostFunctor {
50    template <typename T>
51    bool operator()(const T* const x, T* residual) const {
52      residual[0] = 10.0 - x[0];
53      return true;
54    }
55  };
```

: CostFunctor 구조체 정의 -> 하나의 residual이 된다.

```
double x = 0.5;
const double initial_x = x;

// Build the problem.
Problem problem;

// Set up the only cost function (also known as residual). This uses
// auto-differentiation to obtain the derivative (jacobian).
CostFunction* cost_function =
    new AutoDiffCostFunction<CostFunctor, 1, 1>(new CostFunctor);
problem.AddResidualBlock(cost_function, nullptr, &x);
```

: Problem class, Set Cost Function Class

```
class CostFunction {
 public:
  virtual bool Evaluate(double const* const* parameters,
                        double* residuals,
                        double** jacobians) = 0;
  const vector<int32>& parameter_block_sizes();
  int num_residuals() const;
```

```
AutoDiffCostFunction(CostFunctor* functor,
                     int num_residuals,
                     ownership = TAKE_OWNERSHIP);
```

Reference : http://ceres-solver.org/nnls_tutorial.html#hello-world
https://lee-jaewon.github.io/deep_learning_study/Lec03(TensorFlow)/

3) HelloWorld Tutorial – excute

-- Helloworld.cc와 CMakeLists를 빌드한 ceres-solver 폴더에 배치한 후 build해준 후 실행한다.

https://github.com/ceres-solver/ceres-solver/blob/master/examples/helloworld.cc

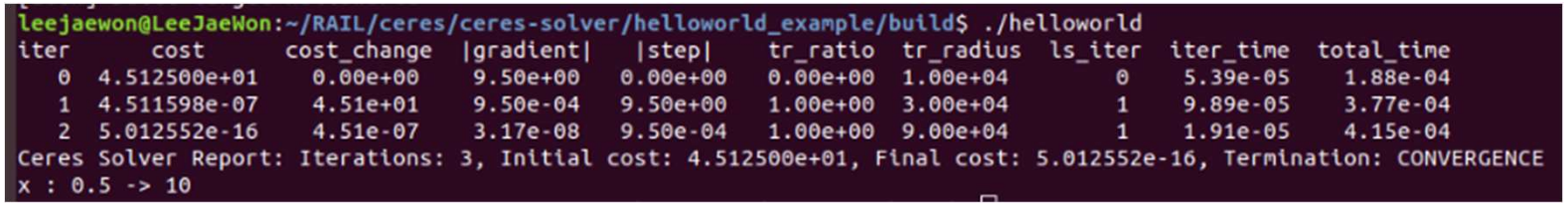
```
cmake_minimum_required(VERSION 2.8)

project(helloworld)

find_package(Ceres REQUIRED)
include_directories(${CERES_INCLUDE_DIRS})

# helloworld
add_executable(helloworld helloworld.cc)
target_link_libraries(helloworld ${CERES_LIBRARIES})
```

-- Result



Reference : http://ceres-solver.org/nnls_tutorial.html#hello-world