The following materials have been collected from the numerous sources such as Stanford CS106 and Harvard CS50 including my own and my students over the years of teaching and experiences of programming. Please help me to keep this tutorial up-to-date by reporting any issues or questions. Please send any comments or criticisms to idebtor@gmail.com. Your assistances and comments will be appreciated.

# Problem Set 1 - HelloWho

## Table of Contents

# Introduction

In this first problem set, we set up our programming environment on your computer as well as joining Piazza SNS service. Also we write the first program that accepts input from the console and process the input as requested.

# Getting Started

After you read head-outs and do the following:

● Join Piazza.

● Install software packages (MinGW/MSYS) – **32bit version** recommended

    ■ Good guide:
      http://holawang.blogspot.kr/2014/02/g++-installing-g++-at-windowsmingw-or.html

    ■ Change (add) the following two folders into user's PATH environment variable:
      `C:\MinGW\bin`
      `C:\MinGW\msys\1.0\bin`

    ■ To check your installation, do the following in cmd windows or in PowerShell;
      `> g++ --version`

    ■ If you encounter the following error message during the first compilation:
      c:/mingw/bin/../lib/g++/mingw32/5.3.0/../../../../mingw32/bin/ld.exe: cannot find -lpthread

      http://stackoverflow.com/questions/39185248/eclipse-mingw-c-cannot-find-lpthread

● Install **Atom**.

- Create a folder that you can access easily.
  For example:  **~/Dropbox/nowic** or **~/Desktop/nowic**

- Download **pset01.zip** from Piazza, and unzip them at **nowic** folder.

  Now your development folder structures should look like the following:

  ```
  ~/nowic/include
  ~/nowic/lib
  ~/nowic/src
  ~/nowic/psets/pset01/pset01.pdf, hellox.exe, names.txt
  ```

Files provided:

```
~/nowic/psets/pset01/pset01.pdf          # this file
~/nowic/psets/pset01/hellox.exe          # a solution to compare your work
~/nowic/psets/pset01/names.txt           # a list of names used in Step 5.
```

# Overview:  Hello <Who>!
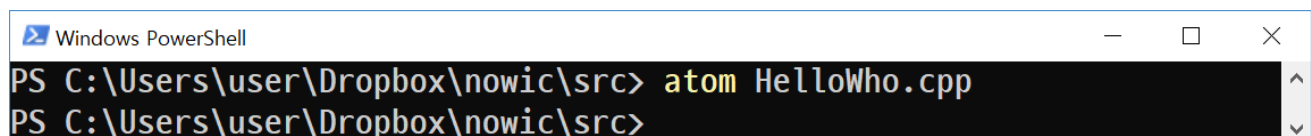
In this problem set, we want to learn

- g++ compilation and execution in a command line

- Processing the command-line arguments passing through **main(int argc, char *argv[])**

- Handling i/o at console using **iostream, cout, cin, getline()**

In this program, you will write **HelloWho.cpp** that prints "Hello <Who>!" where <Who> is a name.

- Your program must accept a single command-line argument as a name.

- If the user does not give a name  as a command-line argument, then you keep on asking the user to enter a name in your program one at a time.

- The user hits an <Enter> without entering a name, you print "Hello World!" and quit the program.

## Step 1. Create a source file: HelloWho.cpp

Write your source program, **~/nowic/src/HelloWho.cpp**, that prints "Hello World!" on the console.  First, use **Atom** at a console as shown below:



Then enter the following code in **HelloWho.cpp** such that it resides in **~/nowic/src** folder.

```
/*
HelloWho.cpp by idebtor@gmail.com
It prints "Hello World!" on the console or "Hello" with a given name.

To build the program which generates hellowho.exe:
> g++ HelloWho.cpp -o hello

To run the program without a command line argument:
> ./hello
> Enter a name:
```

```
    To run the program with a command line argument:
    > ./hello Dr. John Kim
    > Hello Dr. John Kim!

    02/10/19: Created
    */
    #include <iostream>
    #include <string>
    using namespace std;

    int main(int argc, char *argv[]) {
        // Use setvbuf() to prevent the output from buffered on console.
        // setvbuf(stdout, NULL, _IONBF, 0);

        for (int i = 0; i < argc; i++)
            cout << argv[i] << endl;
    }

    cout << "Hello World!"

        // Use system("pause") to prevent the terminal from disappearing
        // as soon as the program terminates such as Visual Studio sometimes.
        // system("pause");
        return EXIT_SUCCESS;
    }
```
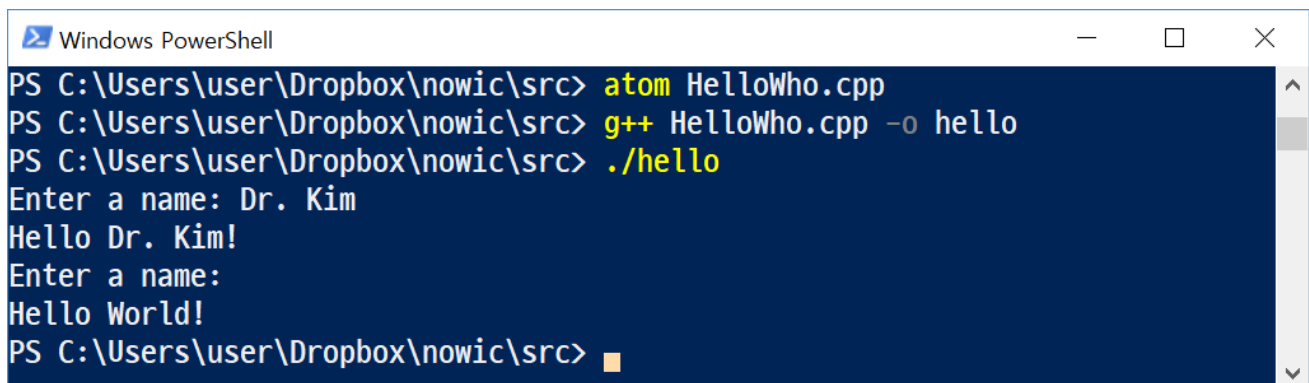
# Step 2. (Build = Compilation + Linking) → Executable

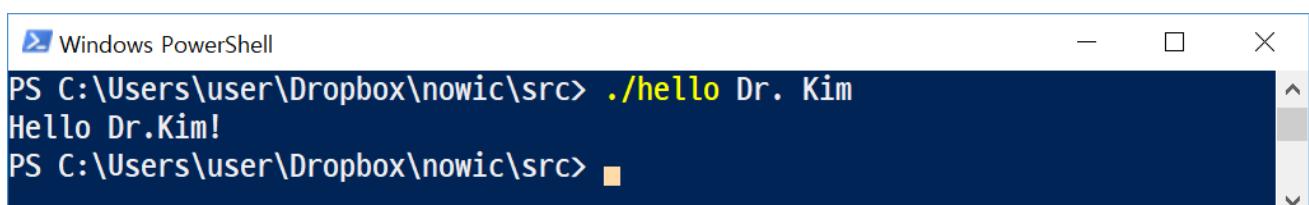Once you have a source file, you may compile it to make an executable.

● The first command line shown below produces **hello.exe** at the current folder.

● The second line runs the program. You must have **./** in front of **hello** if you are using MS Powershell as a console, otherwise you may not need. It tells Powershell that your command (or your executable ~.exe file) exists at the current folder.



# Step 3. Command-line arguments

Now we want to use the command line to pass a list of names such that your program greets them individually as shown below:

This part of the program needs to accept a command-line argument.  Then, you need to declare main with:

```
int main(int argc, char *argv[])
```

The first argument **argc** has the number of arguments in the command line.  For example, if a command line is set as shown below
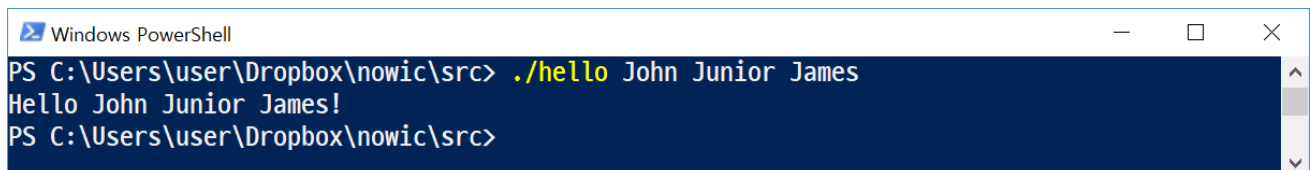
```
➤ ./hello John Junior James
```

Then argc and argv are set as shown below by the system automatically.

```
argc = 4
argv[0] = "./hello"
argv[1] = "John"
argv[2] = "Junior"
argv[3] = "James"
```

Recall that **argv** is an "array" of strings. You can think of an array as row of gym lockers, inside each of which is some value (and maybe some socks). In this case, inside each such locker is a string. To open (i.e., "index into") the first locker, you use syntax like **argv[0],** since arrays are "zero-indexed." To open the next locker, you use syntax like **argv[1].** And so on. Of course, if there are **n** lockers, you'd better stop opening lockers once you get to **argv[n - 1],** since **argv[n]** doesn't exist! (That or it belongs to someone else, in which case you still shouldn't open it.)  In other words, just as argv is an array of strings, so is a **string** an array of chars. And so you can use square brackets to access individual characters in strings just as you can individual strings in **argv.**
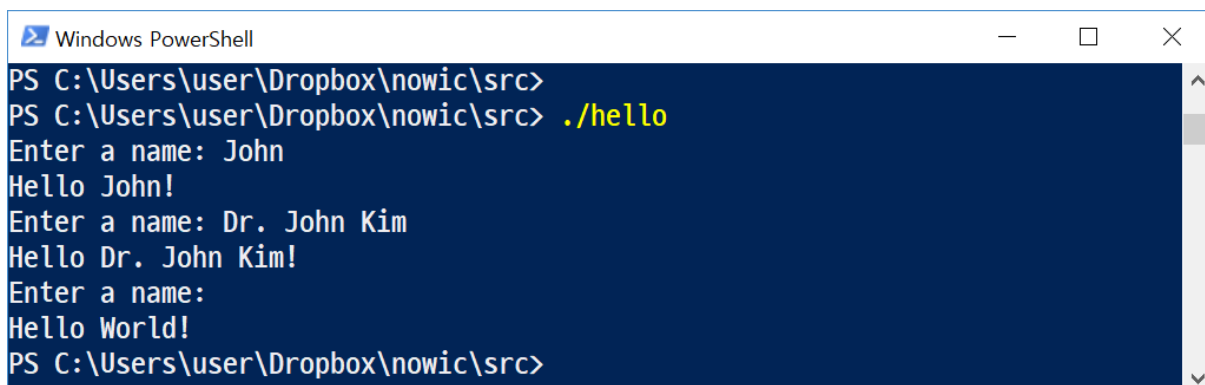
Change your program such that it acts like below:

```
Windows PowerShell                                          —    □    ×
PS C:\Users\user\Dropbox\nowic\src> ./hello John Junior James
Hello John Junior James!
PS C:\Users\user\Dropbox\nowic\src>
```

# Step 4.  Getting a name from the user

Instead of printing "Hello World!" when no names are given in the command-line, ask the user to enter a name repeatedly until he/she enters nothing or enter.  Eventually the user enters <Enter>, quit the program with "Hello World!".  The sample run is shown below:
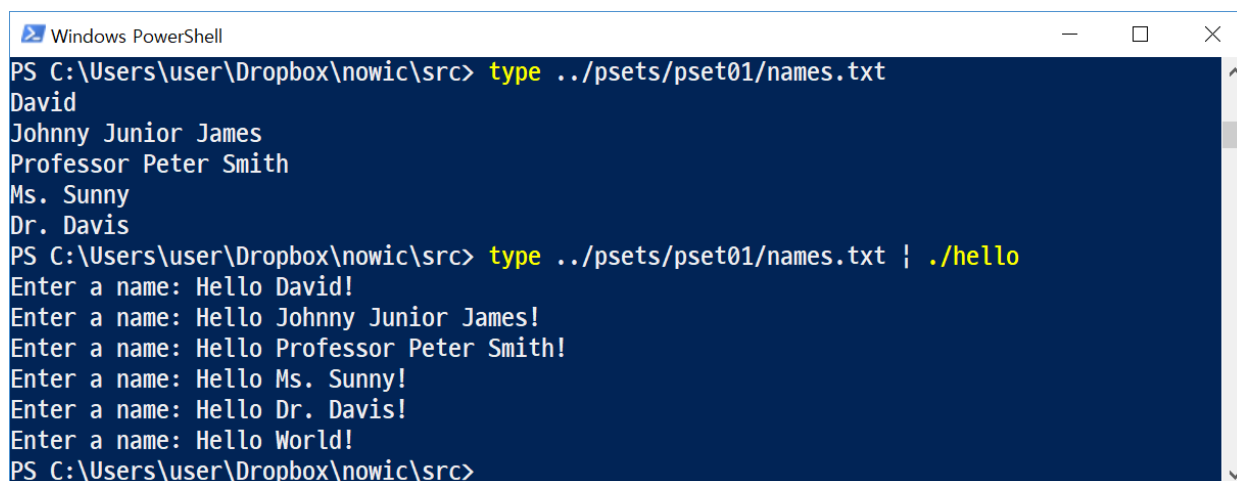
```
Windows PowerShell                                          —    □    ×
PS C:\Users\user\Dropbox\nowic\src>
PS C:\Users\user\Dropbox\nowic\src> ./hello
Enter a name: John
Hello John!
Enter a name: Dr. John Kim
Hello Dr. John Kim!
Enter a name:
Hello World!
PS C:\Users\user\Dropbox\nowic\src>
```

**Compilation** refers to the processing of source code files (.c, .cc, or .cpp) and the creation of an 'object' file. This step doesn't create anything the user can actually run. Instead, the compiler merely produces the machine language instructions that correspond to the source code file that was compiled.

**Linking** refers to the creation of a single executable file from multiple object files. In this step, it is common that the linker will complain about undefined functions (commonly, main itself). During compilation, if the compiler could not find the definition for a particular function, it would just assume that the function was defined in another file.

# Step 5. Run your program using 'Pipe'

Your program should be executed by redirection and pipe as well. This part is to introduce you to the concepts of 'pipe'.  We will use it later on.

```
Windows PowerShell                                                    —    □    ×
PS C:\Users\user\Dropbox\nowic\src> type ../psets/pset01/names.txt
David
Johnny Junior James
Professor Peter Smith
Ms. Sunny
Dr. Davis
PS C:\Users\user\Dropbox\nowic\src> type ../psets/pset01/names.txt | ./hello
Enter a name: Hello David!
Enter a name: Hello Johnny Junior James!
Enter a name: Hello Professor Peter Smith!
Enter a name: Hello Ms. Sunny!
Enter a name: Hello Dr. Davis!
Enter a name: Hello World!
PS C:\Users\user\Dropbox\nowic\src>
```

# Submitting your solution

- Include the following line at the top of your every source file with your name signed.
  **On my honour, I pledge that I have neither received nor provided improper assistance in the completion of this assignment. Signed: _____**
- Make sure your code **compiles** and **runs** right before you submit it.  Every semester, we get dozens of submissions that don't even compile.  Don't make "a tiny last-minute change" and assume your code still compiles. You will not receive sympathy for code that "almost" works.
- If you only manage to work out the Project partially before the deadline, you still need to turn it in. However, don't turn it in if it does not compile and run.
- Place your source files in the folder you and I are sharing.
- After submitting, if you realize one of your programs is flawed, you may fix it and submit again as long as it is **before the deadline**. You will have to resubmit any related files together, even if you only change one.  You may submit as often as you like.  **Only the last version** you submit before the deadline will be graded.

## Files to submit

- Submit your source file **HelloWho.cpp** on time in the **hw1 folder** in Piazza.
  Follow the TA's guideline when you turn in your file(s) since students from two sections are using the same file folder.  Otherwise, there will be a penalty.
  Remember that your file submitted is kept with the time stamped.

## Due and Grade points

- Due: 11:55 pm, Wednesday, March 6, 2019
- Grade: 2 points

# Option: Using mintty console for Mingw/msys

Instead of using cmd or powershell, you may use the better terminal emulators available. You may open a console with the following commands. Create a shortcut for msys.bat.

```
cd c:/mingw/msys/1.0/msys.bat       # change folder
./msys.bat                          # run bat file
```

● As shown below, create a file at **c:/mingw/msys/1.0/home/user/.profile**

```
atom .profile                       # create .profile
./msys.bat                          # run bat file
```

```
# Created by idebtor@gmail.com - 02/02/19
# This .profile is to be placed in C:\MinGW\msys\1.0\home\user and
#     set Target in the shortcut C:\MinGW\msys\1.0\msys.bat --rxvt
# For mintty
#     set Target in the shortcut C:\MinGW\msys\1.0\msys.bat --mintty

alias ls='ls -aFh --color=auto'
alias ll='ls -alkF'
alias rm='rm -i'
alias c='clear'
alias h='history'

echo c:/minGW/msys/1.0$(pwd)/.profile

# Setting my dev folder as a startup folder of msys.
HOME="/c/users/${LOGNAME}/DropBox"
cd $HOME

# @$(hostname) may be added, if necessary, after $(whoami)
PS1='$(whoami) $(pwd -W)> '
```

● **Additionally,** you may use **mintty** terminal emulator instead of the default Windows.

■ **Install mintty.** This can be done either at the MinGW command line or via the MinGW Installation Manager.
From the MinGW command line, run **mingw-get install mintty.**

```
mingw-get install mintty.
```

■ **Configure MSYS.** Next, you need to edit the shortcut you use to run the MinGW shell. Right-click the msys short-cut and select Properties. Wherever you find your link, the Target should be "C:\MinGW\msys\1.0\msys.bat", potentially with some command-line arguments following it. If an argument selecting a shell is already present (such as --rxvt), remove it.
Finally, add **--mintty** as the **first argument** after "C:\MinGW\msys\1.0\msys.bat".

Now, when launching the MinGW shell through that short-cut, mintty will be used instead of the Windows terminal emulator.  Refer to the complete instructions in the following:
https://sites.google.com/site/axusdev/tutorials/installminttyformsys