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

Getting Started

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

From GitHub, get https://github.com/idebtor/nowic repository in your computer. Keep this repository as "read-only".   Copy them into your own repository or development folders in some place you easily access them. They should look like the following:

```
~/nowic/psets/pset01/pset01.pdf          # this file
~/nowic/psets/pset01/hellox.exe          # a solution to compare your work for Windows
~/nowic/psets/pset01/hellox.out          # a solution to compare your work for macOS
~/nowic/psets/pset01/names.txt           # a list of names used in Step 2.
```

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** as shown below:

```cpp
/*
* file: HelloWho.cpp
* It prints "Hello World!" or "Hello" with a given name.
* The completed code should work as shown below. ">" is a prompt of the console.
*
* To run the program without a command line argument:
*    > ./hello
*    > Enter a name: John Lee
*    > Hello John Lee!
*    > Enter a name: Peter Kim
*    > Hello Peter Kim!
*    > Enter a name:<Enter>
*    > Hello World!
*    >
*
* To run the program with a command line argument:
*    > ./hello John Peter James David Kim
*    > Hello John!
*    > Hello Peter!
*    > Hello James!
*    > Hello David!
*    > Hello Kim!
*    > Hello World!
*    >
*
*    02/10/20: Created, idebtor@gmail.com
*/

#include <iostream>
#include <sstream>
#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] << "!\n";

cout << "Hello World!\n";

    // Use system("pause") to prevent the terminal from disappearing
    // as soon as the program terminates as in Visual Studio sometimes.
    // system("pause");
    return EXIT_SUCCESS;
}
```

This program, HelloWho.cpp, is supposed to work as described in the comment section at the top of the file. The program works differently depending on whether or not there is a give command-line argument. If there is no command-line argument except the executable itself (or hello.exe), it should ask the user to enter a name. Once the user enters a name, print "Hello" with the name. Otherwise finish the program with "Hello World!".

If there are some arguments given, then assume that names are given. In the code, you use a loop statement to print them all and exit the program with "Hello World!".

Hint 1: Command-line arguments

You may review Lecture Notes about command-line arguments available at: **nowic/ArgcArgv.md.**

Now we want to use the command line to pass a list of names such that your program greets them individually. 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
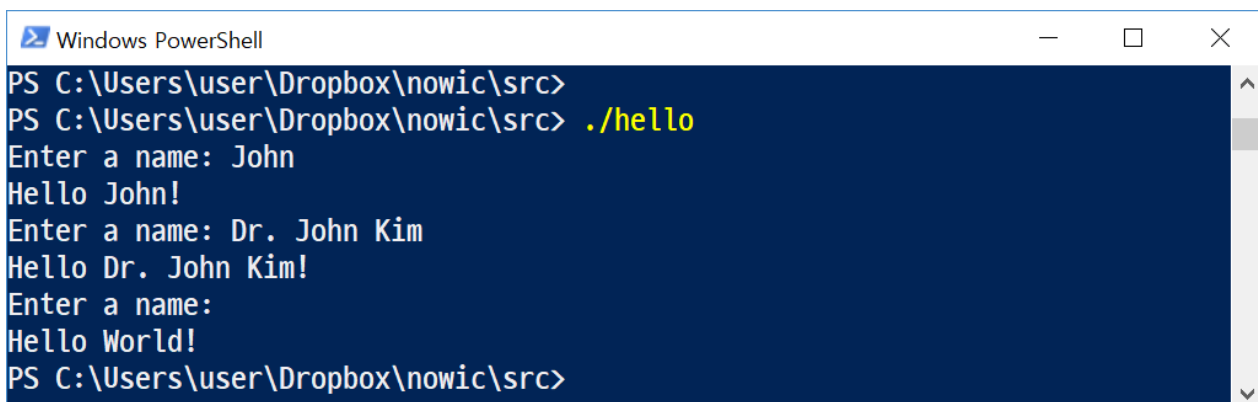
```
./hello John Junior James
```

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.**

Hint 2. Getting a name from the user

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:

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 2: Run your program using 'Pipe'

This part is to introduce you to the concepts of **'pipe'**. Your program should be executed by redirection and **pipe** as well. You may use **cat** instead of **type** in Mac and Linux.

If this part does not work, you must fix your code to work as shown below:

```
PS C:\GitHub\nowicx\src> g++ hello.cpp -o hello
PS C:\GitHub\nowicx\src> type ../psets/pset01hello/names.txt | ./a
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 Prof. Kim!
Enter a name: Hello Handong!
Enter a name: Hello Peter!
Enter a name: Hello World!
계속하려면 아무 키나 누르십시오 . . .
PS C:\GitHub\nowicx\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: _____   Student Number: _____

- 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 **pset1 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, Saturday, March 7, 2019
- Grade:
    - Step 1: 1 point
    - Step 2: 1 point