Tutorial 2
Get familiar with VS Code & Makefile
and
String Applications

Sep. 19, 2022

Lai Wei (USTF)

(SDS, 120090485@link.cuhk.edu.cn)

Objectives today

1. Some background information about this course.
2. Two concepts clarification
3. Get familiar with some common operations of string (e.g., substr), **which will be important in your Assignment 1!**
4. Three examples with string operations:
   1. Palindromes 回文数
   2. Acronym 首字母缩略词
   3. Pig Latin 儿童隐语
5. Learn to use VS Code and Makefile to run the examples above.
6. Q&A time: make sure you can run the C++ code on your computer now! (If you can't, solve it today!)

1. Background

- The course changed a lot in this semester. Now, you can:
- Use Qt Creator to write code. (Recommended)
  - Most examples in our textbook has implementations with the Stanford Library.
  - It is difficult to embed the Library into other IDEs and there is a blank or empty project provided with the Library already installed.
  - So if you want to use Stanford Library, you probably need to use Qt.
- Use VS Code to write code. (Highly Recommended)
  - Familiar, Light-weight, easy to use.
  - Learn to use command lines and Makefile.
  - Allow writing and running single .cpp file without create a heavy “project”.
  - Our assignments are individual problems (individual files), VS Code is enough to use.
- Use other IDEs like Visual Studio or CLion. (Optional)

1. Background

- A fact:
- Our commonly used “Python” is mainly implemented in C language. (https://github.com/python/cpython)
- C/C++ language is more complicated than Python. Python hides all complicated grammar and details in C/C++, providing you an easy-to-use programming language.
- **Why more complicated? Designed for high performance.**
- But in this course, we are learning C++. It is common for you to feel more difficult than CSC1001 (Python). But don’t worry!

2. Concepts

- What is g++?
  - g++ is the traditional nickname of GNU C++, a freely redistributable **C++ compiler** produced by the Free Software Foundation plus dozens of skilled volunteers
  - *(source: http://tinf2.vub.ac.be/~dvermeir/manual/g++/g++faq_1.html)*
- What is Makefile?
  - **Makefile (Script)** is a way of **automating software building procedure** with dependencies.
  - The **make (program)** automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them.
  - **NOTE: make (program)** is only a helper building tool, not the compiler. The core compiler is still g++.
  - *(source: https://www3.nd.edu/~zxu2/acms60212-40212/Makefile.pdf)*

3. Get familiar with some common operations of string

- Two types of String:
- C-style string (char array)
  - `char c[6] = {'h', 'e', 'l', 'l', 'o', '\0'};`
  - `char c[6] = “hello”;`
- C++ std::string
  - `include<string>`
  - `std::string s = "hello";`
- Changing C-style string to std::string
  - `std::string s(c);`
  - `std::string s = std::string(c);`
- Changing std::sting to C-style string
  - `s.c_str();`

Last one be ‘\0’ to make it safe!!

length (6-1) maximum!!!

3. The <cctype> (ctype.h) Interface

This header declares a set of functions to **classify** and **transform individual characters**.

`bool isdigit(char ch)` Determines if the specified character is a digit.
`bool isalpha(char ch)` Determines if the specified character is a letter.
`bool isalnum(char ch)` Determines if the specified character is a letter or a digit.
`bool islower(char ch)` Determines if the specified character is a lowercase letter.
`bool isupper(char ch)` Determines if the specified character is an uppercase letter.
`bool isspace(char ch)` Determines if the specified character is *whitespace* (spaces and tabs).
`char tolower(char ch)` Converts ch to its lowercase equivalent, if any. If not, ch is returned
`char toupper(char ch)` Converts ch to its uppercase equivalent, if any. If not, ch is returned unchanged.

For more, please visit https://cplusplus.com/reference/cctype/

3. The <cstring> (string.h) Interface

This header file defines several functions to **manipulate C strings** and **arrays**.

`void* memcpy(void* dst, const void* src, size_t num)` Copy block of memory
`char* strcat(char* dst, const char* src)` Concatenate strings
`void* memchr (void* ptr, int value, size_t num )` Locate character in block of memory
……

For more, please visit https://cplusplus.com/reference/cstring/

3. Operators on the string Class

- To convert the C++ string objects into C string literals, simply apply the `c_str` method to the C++ string.
- Unlike most languages, C++ allows classes to redefine the meanings of the standard operators. As a result, several string operations, such as + for concatenation, are implemented as operators (overloading).

`str[i]` Returns the ith character of `str`. Assigning to `str[i]` changes that character.
`s1 + s2` Returns a new string consisting of `s1` concatenated with `s2`.
`s1 = s2;` Copies the character string `s2` into `s1`.
`s1 += s2;` Appends `s2` to the end of `s1`.
`s1 == s2` (and similarly for <, <=, >, >=, and !=) Compares to strings lexicographically.
`str.c_str()` Returns a C-style character array containing the same characters as `str`.

3. Operators on the string Class

`str.length()` Returns the number of characters in the string `str`.
`str.at(index)` Returns the character at position `index`; most clients use `str[index]` instead.
`str.substr(pos, len)` Returns the substring of `str` starting at `pos` and continuing for `len` characters.
`str.find(ch, pos)` Returns the first index ≥ `pos` containing `ch`, or `string::npos` if not found.
`str.find(text, pos)` Similar to the previous method, but with a string instead of a character.

4.1. Palindrome

- A *palindrome* is a word that reads identically backward and forward, such as “level” or “noon”.
- Write a C++ program `isPalindrome` that checks whether a string is a palindrome.

```
bool isPalindrome(string str) {
   int n = str.length();
   for (int i = 0; i < n / 2; i++) {
      if (str[i] != str[n - i - 1]) return false;
   }
   return true;
}
```

```
bool isPalindrome(string str) {
   return str == reverse(str);
}
```

we define this function to reverse a string.

- Efficiency vs. Readability

4.2. Acronym

- An *acronym* is a word formed by taking the first letter of each word in a sequence, as in

  `"self-contained underwater breathing apparatus"` → `"scuba"`

- Write a C++ program that generates acronyms, as illustrated by the following sample run:

Acronym

```
Program to generate acronyms
Enter string: not in my back yard
The acronym is "nimby"
Enter string: Federal Emergency
The acronym is "FEMA"
Enter string: Management Agency
```

4.2. Acronym

```
string acronym(string str) {
   string result = "";
   bool inWord = false;
   int nc = str.length();
   for (int i = 0; i < nc; i++) {
      char ch = str[i];
      if (inWord) {
         if (!isalpha(ch)) inWord = false;
      } else {
         if (isalpha(ch)) {
         result += ch;
         inWord = true;
         }
      }
   }
   return result;
}
```

4.3. Translating English to Pig Latin

We describe a C++ program that reads a line of text from the user and then translates each word in that line from English to Pig Latin, a made-up language familiar to most children in the English-speaking world.

In Pig Latin, words are formed from their English counterparts by applying the following rules:

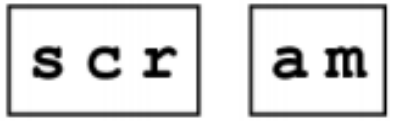
1. If the word contains no vowels (元音), no translation is done, which means that the translated word is the same as the original.

2. If the word begins with a vowel, the function adds the string "way" to the end of the original word.

3. If the word begins with a consonant (辅音), the function extracts the string of consonants up to the first vowel, moves that collection of consonants to the end of the word, and adds the string "ay".

4.3. Translating English to Pig Latin

As an example, suppose that the English word is *scram*. Because the word begins with a consonant, you divide it into two parts: one consisting of the letters before the first vowel and one consisting of that vowel and the remaining letters:

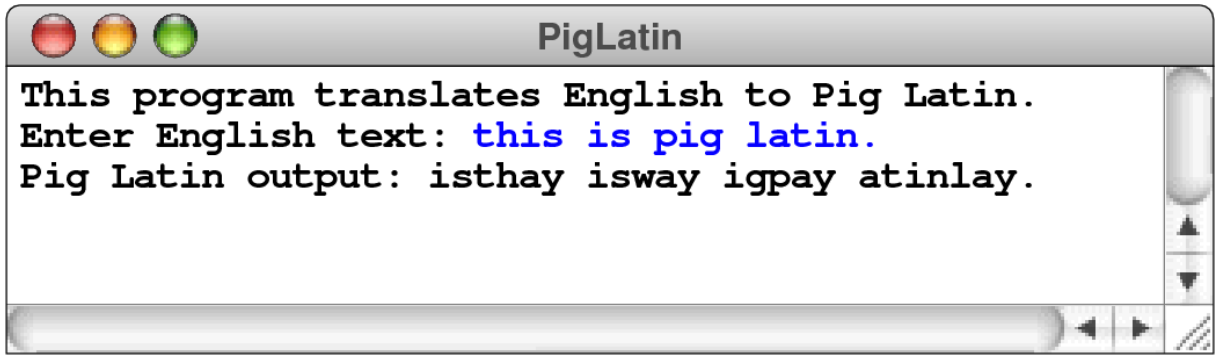


You then interchange these two parts and add ay at the end, as follows:

Thus the Pig Latin word for *scram* is *amscray*. For a word that begins with a vowel, such as *apple*, you simply add way to the end, which leaves you with *appleway*.

4.3. Translating English to Pig Latin

A sample run of the program might look like this:



It is worth taking a careful look at the implementations of *lineToPigLatin* and *wordToPigLatin*. The *lineToPigLatin* function finds the word boundaries in the input and provides a useful pattern for separating a string into individual words. The *wordToPigLatin* function uses *substr* to extract pieces of the English word and then uses concatenation to put them back together in their Pig Latin form. In Chapter 6, you will learn about a more general facility called a token scanner that divides a string into its logically connected parts.

5.1. Run the code via command lines (terminal)

- If you don't know how, or fail to set the VS Code makefile extension (settings.json), you can **ALWAYS** use **COMMAND LINES** to compile and run your C++ code.
- Only pre-requisite: you can run "make --version" and "g++ --version" in the command lines.

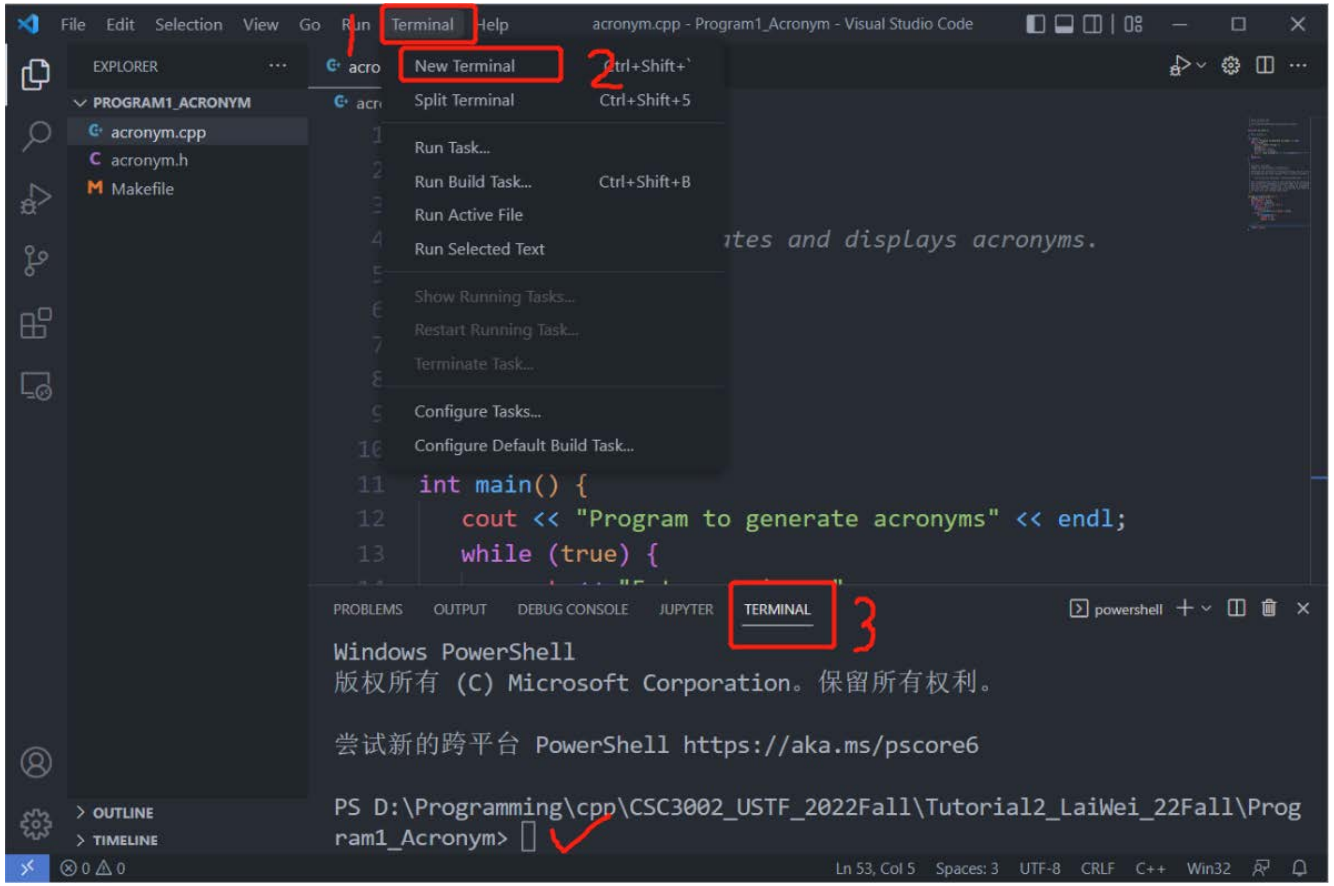
```
PS C:\Users\30785> make --version
GNU Make 3.81
Copyright (C) 2006  Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

This program built for i386-pc-mingw32
PS C:\Users\30785> g++ --version
g++.exe (x86_64-posix-seh-rev0, Built by MinGW-W64 project) 7.3.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

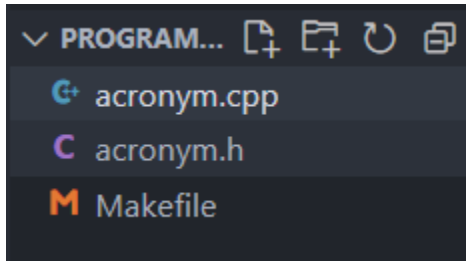
5.1. Run the code via command lines

a) Compile by **pure** command lines, with “g++” compile command

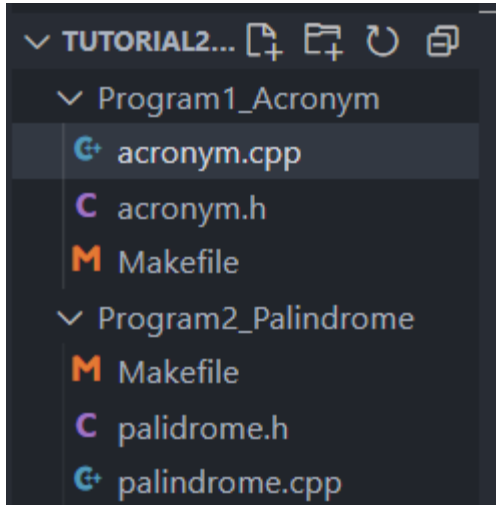
- Open a terminal in the current code folder.



Correct: Makefile is in current program folder workspace, no path problem.



Wrong: Makefile in NOT in program folder workspace, may bring relative-path problem.



5.1. Run the code via command lines
a) Compile by **pure** command lines, with “g++” compile command

- In the terminal, type compile command:
- “g++ -std=c++17 <your source .cpp files> -o <output filename>”
- E.g. “g++ -std=c++17 helloworld.cpp foo.cpp -o helloworld”

- Then run the executable program:
- different terminal have different calling method, maybe:
  - “./<filename>”, “<filename>” (macOS system)
  - “./<filename>.exe”, “<filename>.exe” (Windows system)

5.1. Run the code via command lines

a) Compile by **pure** command lines, with “g++” compile command

```
尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial2_LaiWei_22Fall\Program1_Acronym> g++ -std=c++17 acronym.
cpp -o acronym.exe
PS D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial2_LaiWei_22Fall\Program1_Acronym> ./acronym.exe
Program to generate acronyms
Enter string:
```

compile

run

- Use <ctrl>(command) + C to exit the program.

5.1. Run the code via command lines
b) Compile by **pure** command lines, with “makefile”.

- Write your own “Makefile” script, or use the template given. (If you use Prof Kinley’s template, remember to change two names!)
- In the terminal, type make command:
- “make” (or “mingw32-make.exe”, if you use Qt’s make tool)

```
PROGRAM = \
    helloworld

OBJECTS = \
    helloworld.o \
    foo.o \
```

- Then run the executable program:
- different terminal have different calling method, maybe:
  - “./<filename>”, “<filename>” (macOS system)
  - “./<filename>.exe”, “<filename>.exe” (Windows system)

5.1. Run the code via command lines
b) Compile by **pure** command lines, with “makefile”.

```
尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial2_LaiWei_22Fall\Program1_Acronym> make
g++ -c -std=c++17 acronym.cpp -o acronym.o
g++ -std=c++17 acronym.o -o acronym
PS D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial2_LaiWei_22Fall\Program1_Acronym> ./acronym.exe
Program to generate acronyms
Enter string:
```

make

run

- Use <ctrl>(command) + C to exit the program.

5.1. Run the code via command lines
b) Compile by **pure** command lines, with “makefile”.

- One useful tip: Use “↑” “↓”(Up/down arrows on the keyboard) to view command history.
- Two useful tips: Use “Tab” for Automatic completion the command / filename
- (Demo)

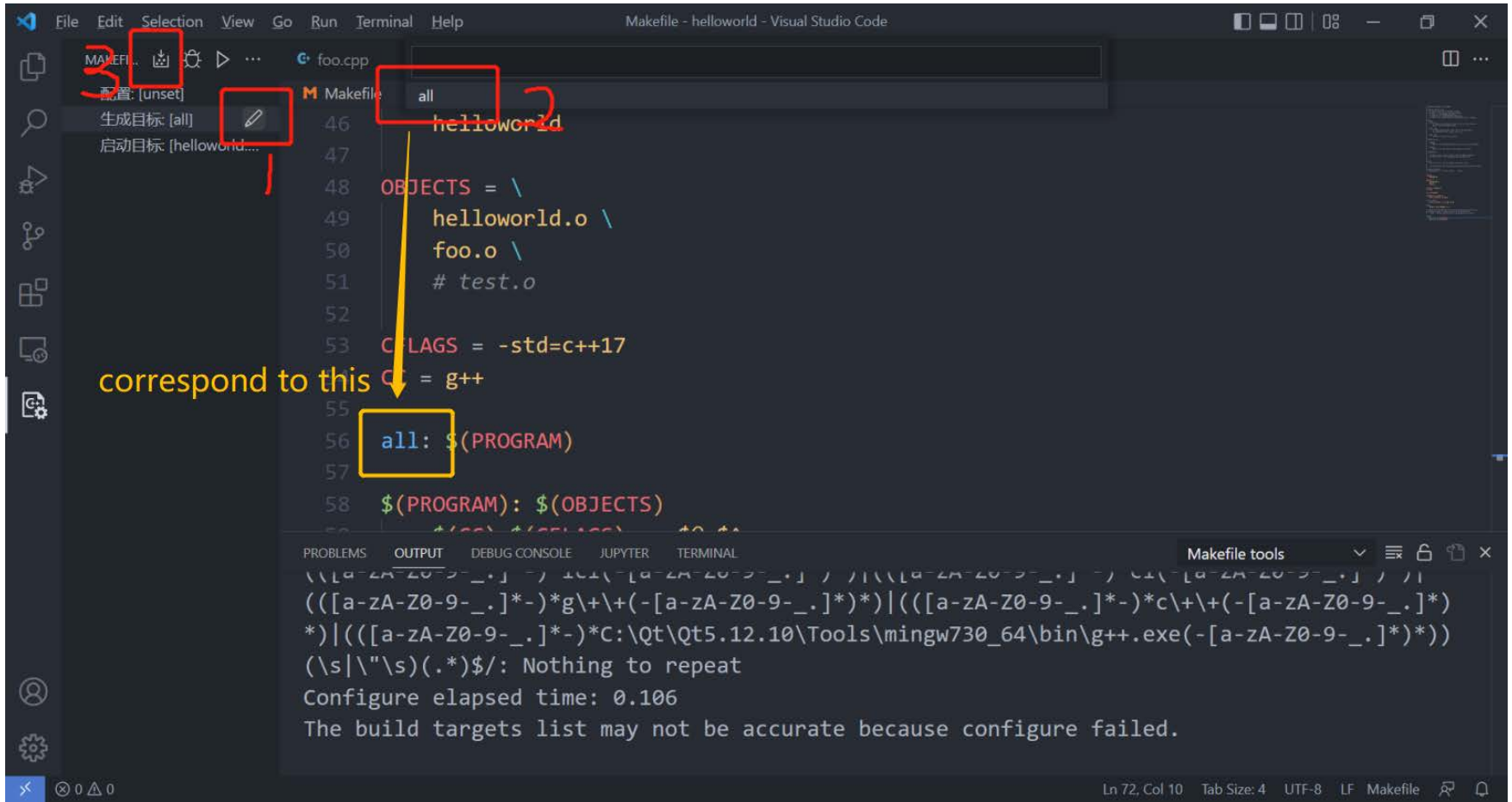
5.2. Run the code via VS Code make-extension

- A fact behind the make-extension: **it automatically generates commands in terminal** for you after you **pressing the "build" "run" button**. In the first time you press the button, it automatically generates configurations for you (in ".vscode/settings.json").
- If it succussed, you can use the button to compile and run your program.
- But if it failed, you need to manually set the proper path settings (in ".vscode/settings.json").

- But, you can **ALWAYS** use **COMMAND LINES** to compile and run your C++ code.

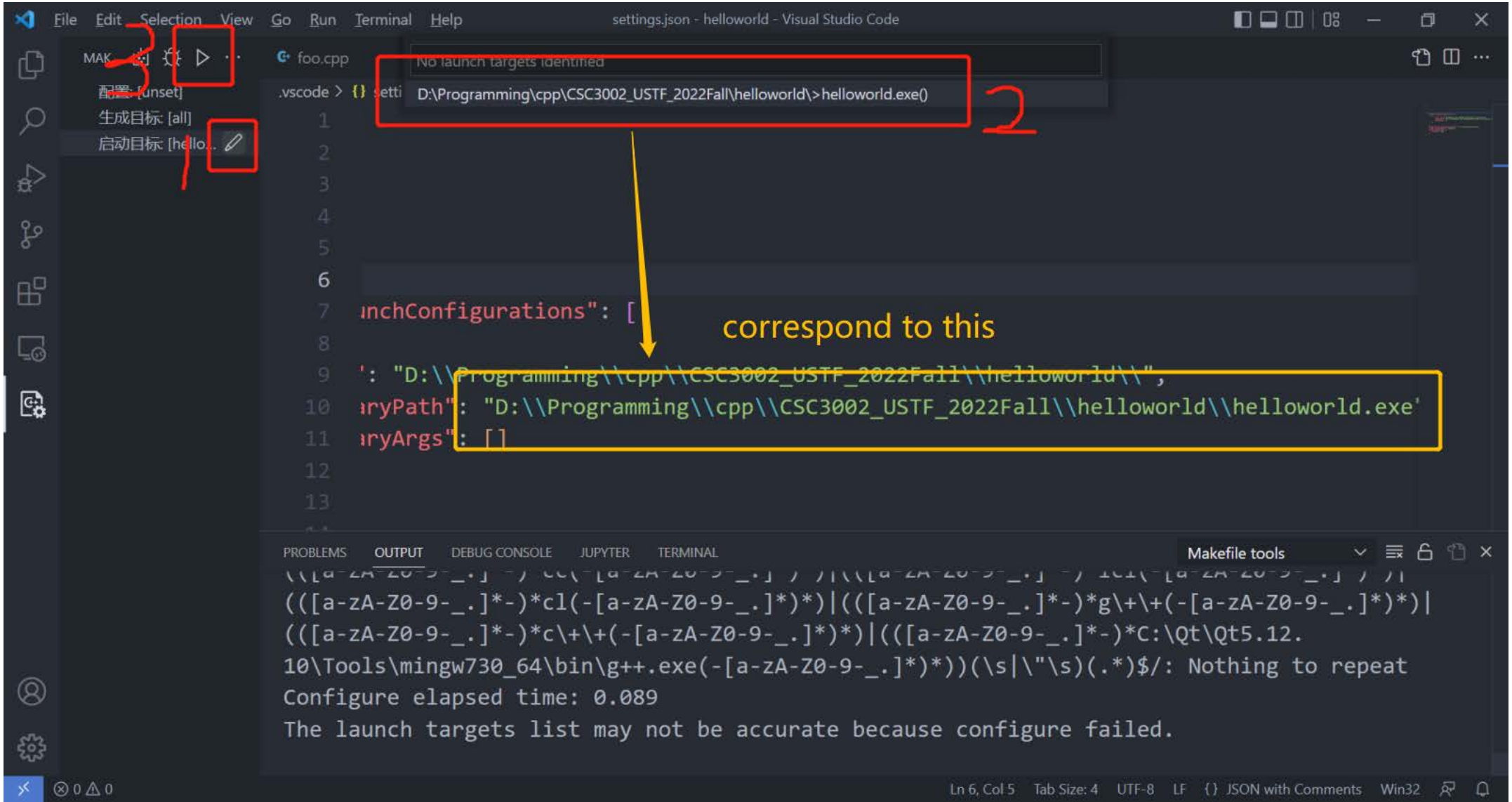
5.2. Run the code via VS Code make-extension

- You can **first directly use the buttons** to compile and run your program without setting path in “settings.json”. The Extension will **automatically detect and generate the settings** for you. If you find the setting path is not correct, view next-next slide for solution.
- Build:



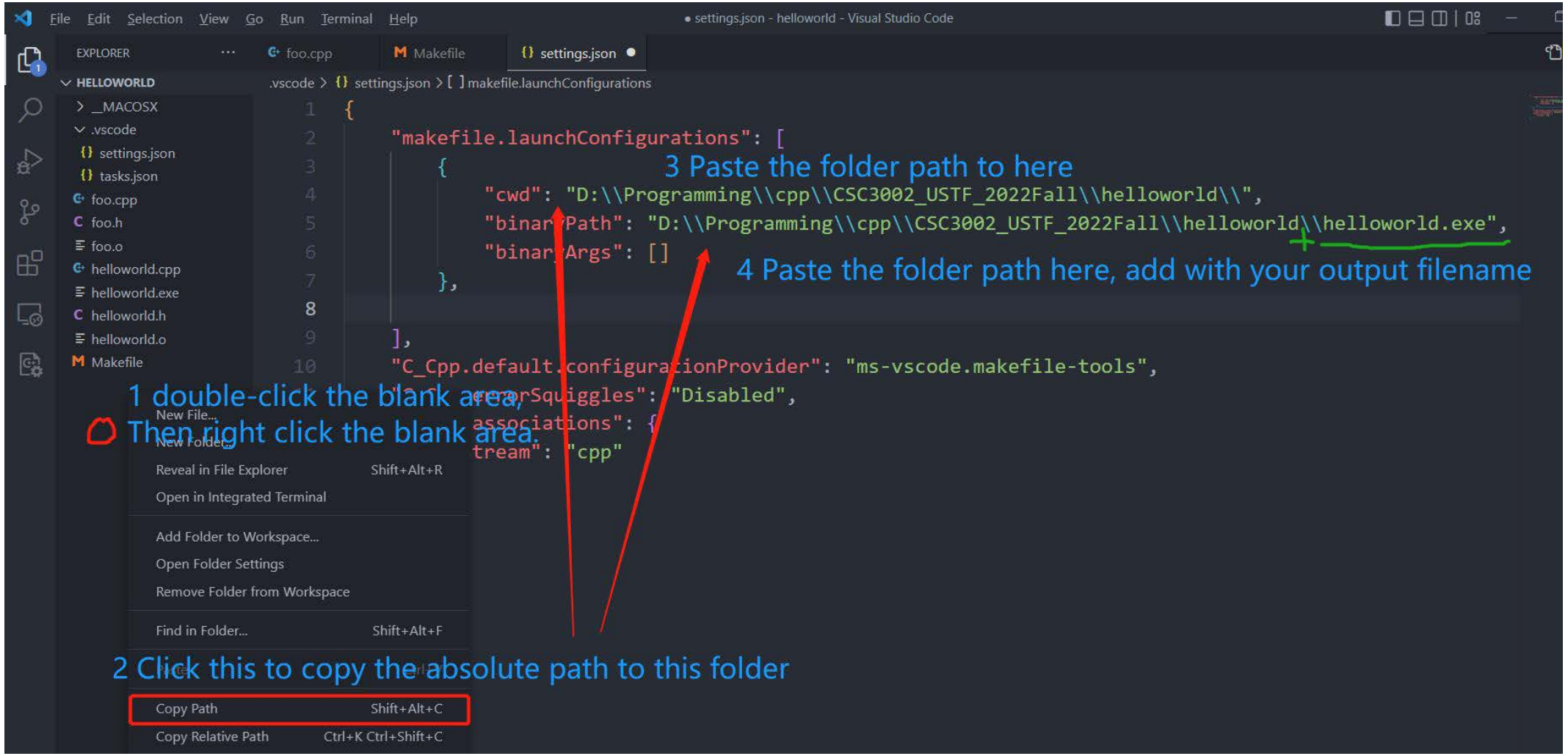
5.2. Run the code via VS Code make-extension

- Run:



5.2. Run the code via VS Code make-extension

- If you find the auto-generated setting path is not correct, you can manually set the path in the “settings.json”:



6. Q & A time

- Thank you for your listening!

- Lai Wei (USTF)
- (SDS, 120090485@link.cuhk.edu.cn)

- Office hour: Friday 10:00-11:00 am, Start-up Zone library L103