

# OOP & data struct

## 1.Introduction & Development of Coding

---

BY SOMSIN THONGKRAIRAT



# What is OOP & data structure

---

Object-oriented programming (OOP)

+

Data structure & Algorithm

≈

**Programming 2**

# What is Object-oriented programming (OOP)

---

Programming paradigm using “OBJECT” concept

รูปแบบหนึ่งของการเขียนโปรแกรมที่มองทุกอย่างเป็น “OBJECT”

Most popular of Programming agreement (2022)

เป็นรูปแบบที่ใช้กันมากที่สุด (2565)

Year

2022



Quarter

3



# Ranking	Programming Language	Percentage (YoY Change)	YoY Trend
1	JavaScript	17.392% (-4.641%)	
2	Python	16.946% (+0.493%)	
3	Go	12.714% (+1.379%)	
4	Java	8.521% (-0.487%)	
5	C++	8.478% (+0.115%)	
6	TypeScript	7.293% (+2.681%)	^
7	C	5.315% (+0.251%)	v
8	C#	3.892% (+0.224%)	^
9	PHP	3.682% (-0.132%)	v
10	Shell	2.540% (-0.030%)	
11	Rust	2.478% (+1.191%)	^
12	Ruby	1.835% (-0.110%)	
13	Swift	1.780% (-0.266%)	v
14	Kotlin	1.298% (+0.259%)	

# What is Data structure & Algorithm

---

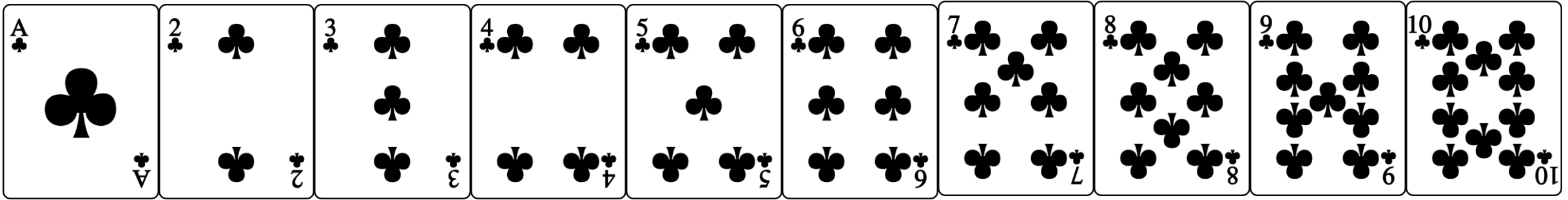
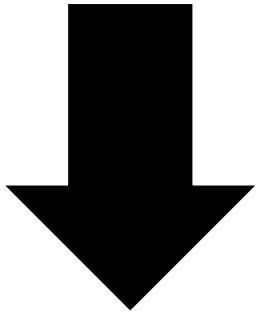
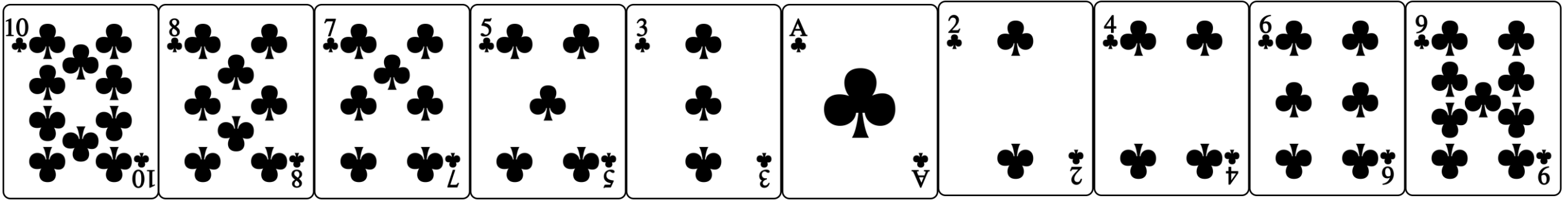
How to store and organize data

How to solve a problem

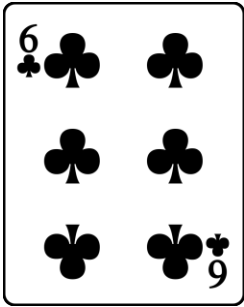
วิธีเก็บข้อมูล

วิธีการทำงานของโปรแกรม

# Example : Sorting



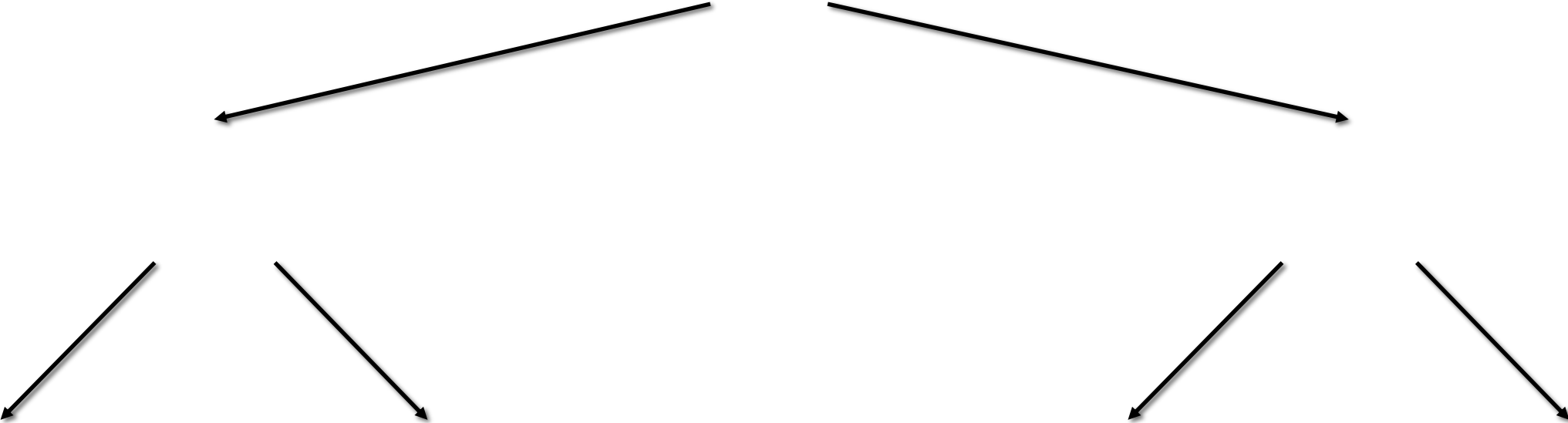
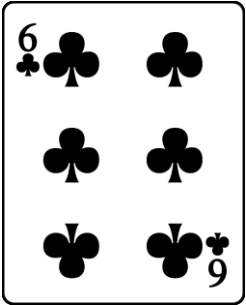
7 Card Here!



--	--	--	--	--	--	--

0	1	2	3	4	5	6
---	---	---	---	---	---	---

Binary tree & [ $<$ ]





# Data structure & Algorithm

---

Data structure -  $>$  Tree

Algorithm -  $>$  Less Than ( $<$ )

# Outline

---

## Object-oriented programming (OOP)

- Midterm Exam

## Data structure & Algorithm

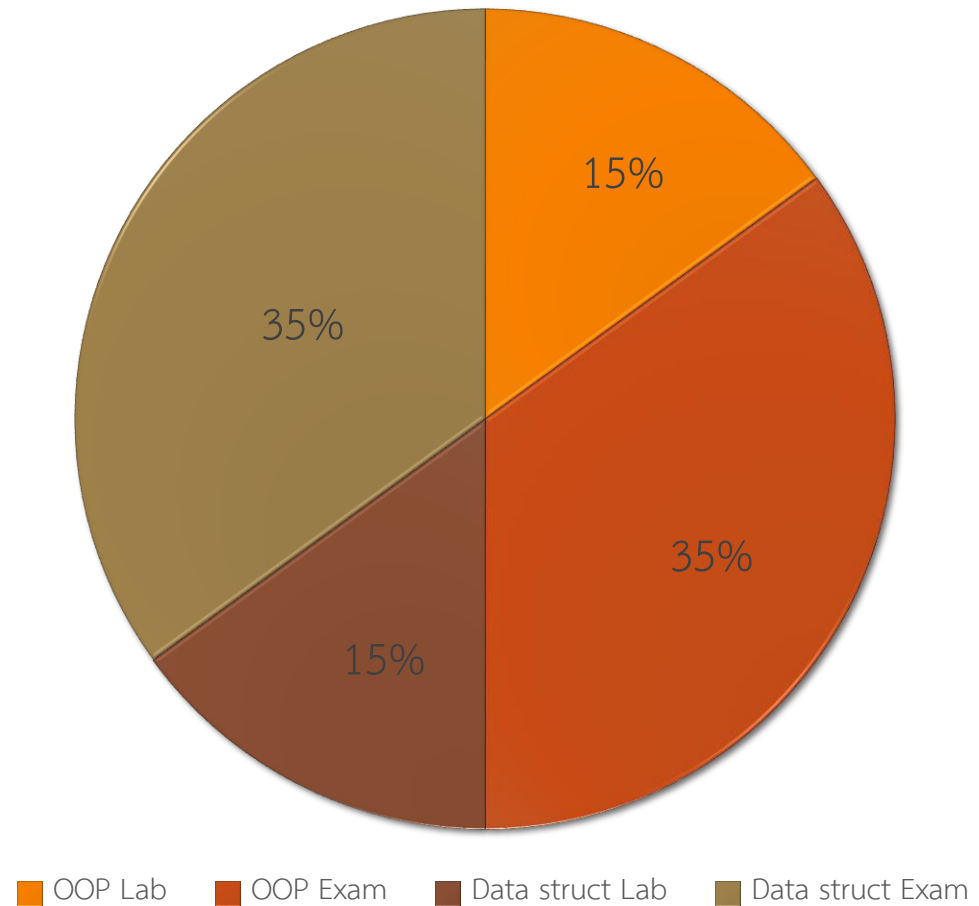
- Final Exam

# Scoring

Lab 30 %

Exam 70%

percentage



# Why

---

Let's know the history, If not OOP!

เรียนรู้จากอดีต ก่อนหน้า OOP

# Non-structural Language

---

Using “jump” or “goto” statement

ภาษาที่ใช้คำสั่ง Jump หรือ Goto

ASM : Assembly language

```
1      mov ax, 5 ; set ax to 5.
2      mov bx, 2 ; set bx to 2.
3      jmp calc ; go to 'calc'.
4  back:
5      jmp stop ; go to 'stop'.
6  calc:
7      add ax, bx ; add bx to ax.
8      jmp back ; go 'back'.
9  stop:
10     ret ; return to operating system.
11
```

# Jump

---

Instruction	Description	Flags tested
JE/JZ	Jump Equal or Jump Zero	ZF
JNE/JNZ	Jump not Equal or Jump Not Zero	ZF
JG/JNLE	Jump Greater or Jump Not Less/Equal	OF, SF, ZF
JGE/JNL	Jump Greater/Equal or Jump Not Less	OF, SF
JL/JNGE	Jump Less or Jump Not Greater/Equal	OF, SF
JLE/JNG	Jump Less/Equal or Jump Not Greater	OF, SF, ZF

# Looping

```
1      mov cx, N
2      jmp bottom
3  top:
4      BODY ; print something
5      dec cx
6  bottom:
7      cmp cx, 0
8      jne top
```

# Non-structural Language

---

## ข้อเสีย

- กำหนดขอบเขตด้วยคำสั่ง jump หรือ goto (ไม่มี parentheses () {} [])
- ยากต่อการใช้ flow control (flow chart)
- ยากต่อการทำงานร่วมกัน!



# Non-structural Language

---

## Disadvantage

- Scope flow with “JUMP” (NO parentheses () {} [])
- Hard to implemented flow control (flow chart)
- lack of collaboration!

# Procedural Oriented Programming

---

concept of calling procedure. Procedures, also known as routines, subroutines or functions

การเขียนโปรแกรมโดยเรียกใช้ scope คำสั่ง หรือ function

FORTRAN, ALGOL, COBOL, Pascal and C.

```
1  #include<stdio.h>
2
3  int multiply_int_int(int a,int b){
4      int c = a * b;
5      return c;
6  }
7
8  float multiply_int_float(int a,float b){
9      float c = a * b;
10     return c;
11 }
12
13 float multiply_float_int(float a,int b){
14     float c = a * b;
15     return c;
16 }
17
18 int main(){
19
20     int a,b;
21     float c,d;
22     a = 1;
23     b = 2;
24     c = 3;
25     d = 4;
26
27     printf("%d\n", multiply_int_int(    a,b));
28     printf("%f\n", multiply_int_float(  a,c));
29     printf("%d\n", multiply_float_int(  d,b)); // what happened!
30
31     return 0;
32 }
```

```
1  #include<stdio.h>
2
3  ✓ int main(){
4
5      char a[] = "Haruhi!Haruhi!Haruhi!Haruhi!Haruhi!Haruhi!";
6      for(int i=7;i<14;i++){
7          printf("%c",a[i]);
8      }
9
10     return 0;
11 }
```

# Object-oriented programming (OOP)

---

Programming paradigm using “OBJECT” concept

รูปแบบหนึ่งของการเขียนโปรแกรมที่มองทุกอย่างเป็น “OBJECT”

# OOP Concept

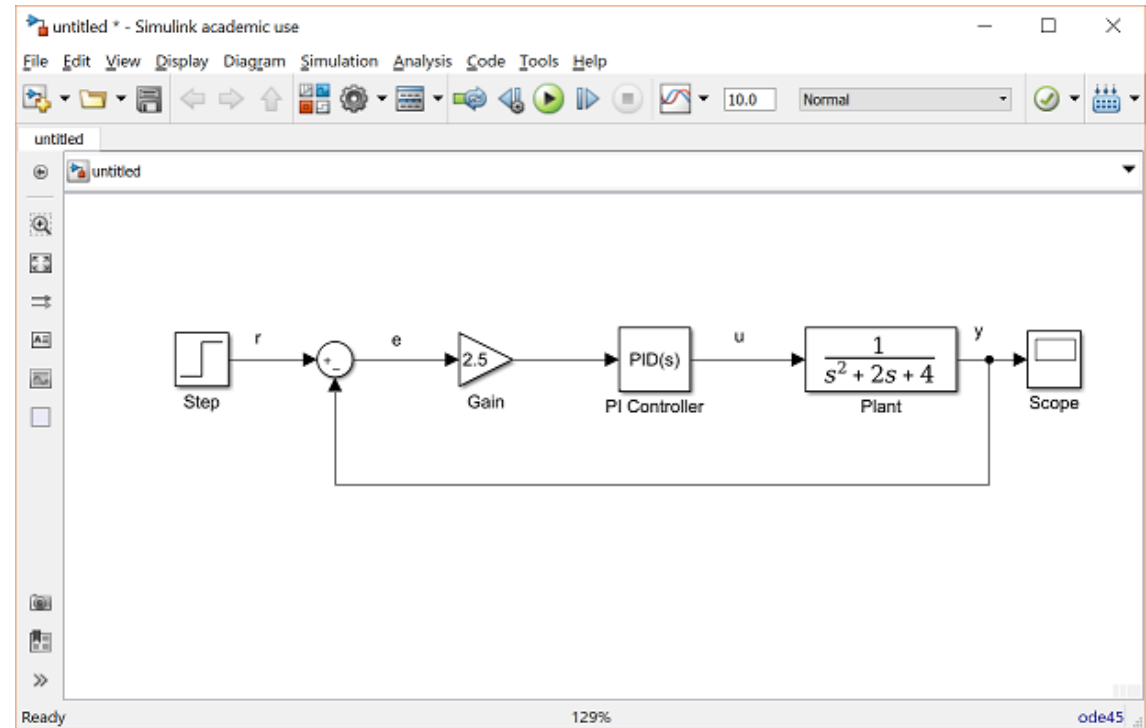
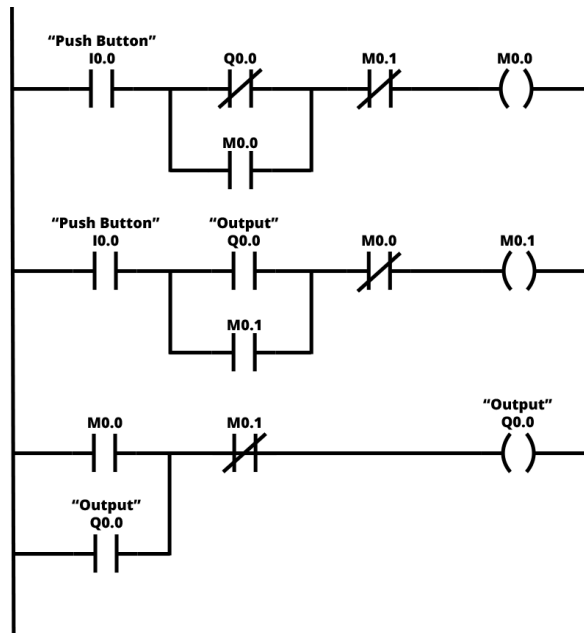
---

- Encapsulation - เก็บตัวแปรและ function ไว้ใน class
- Abstraction - ไม่จำเป็นต้องเขียน function ให้สมบูรณ์
- Inheritance - สามารถนำ code ที่เขียนไว้อยู่แล้วมาใช้ใหม่ได้
- Polymorphism – function มีความยืดหยุ่นปรับตัวได้ตามการใช้งาน

BY C++

# What if not OOP?

- model base programming (Simulink , UE)
- ladder (PLC)



# What if not OOP?

- Concurrent statement (VHDL)
- Quantum Computing (IBM Quantum GATE)

```
library ieee;
use ieee.std_logic_1164.all;
library work;

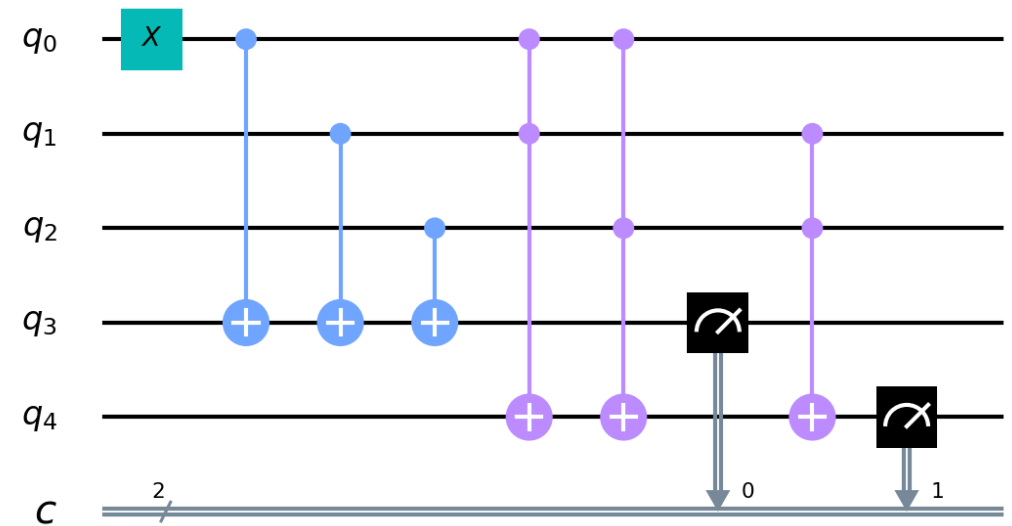
entity top is port (
  in1 : in std_logic;
  out1 : out std_logic);
end top;

architecture beh of top is

begin

  u0: entity work.bottom
    port map (in1 => in1, out1 => out1);

end beh;
```





## Selecting Language (การเลือกใช้)

---

- Depending on purpose จุดประสงค์ของการใช้งาน
- hardware capability ความสามารถของ hardware
- collaboration size จำนวนของผู้ร่วมงานในทีม
- support availability เข้ากับยุคสมัย

## Conclusion (บทสรุป)

---

- History of programming paradigm
- Why we have to study this?
- ประวัติของรูปแบบการเขียนโปรแกรม
- ทำไมต้องเรียนวิชานี้

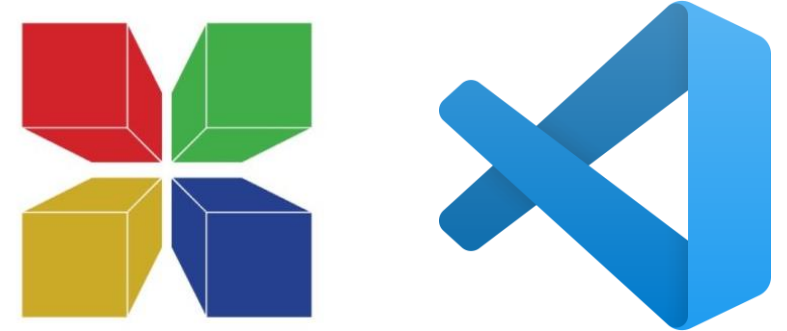
# LAB Section

---

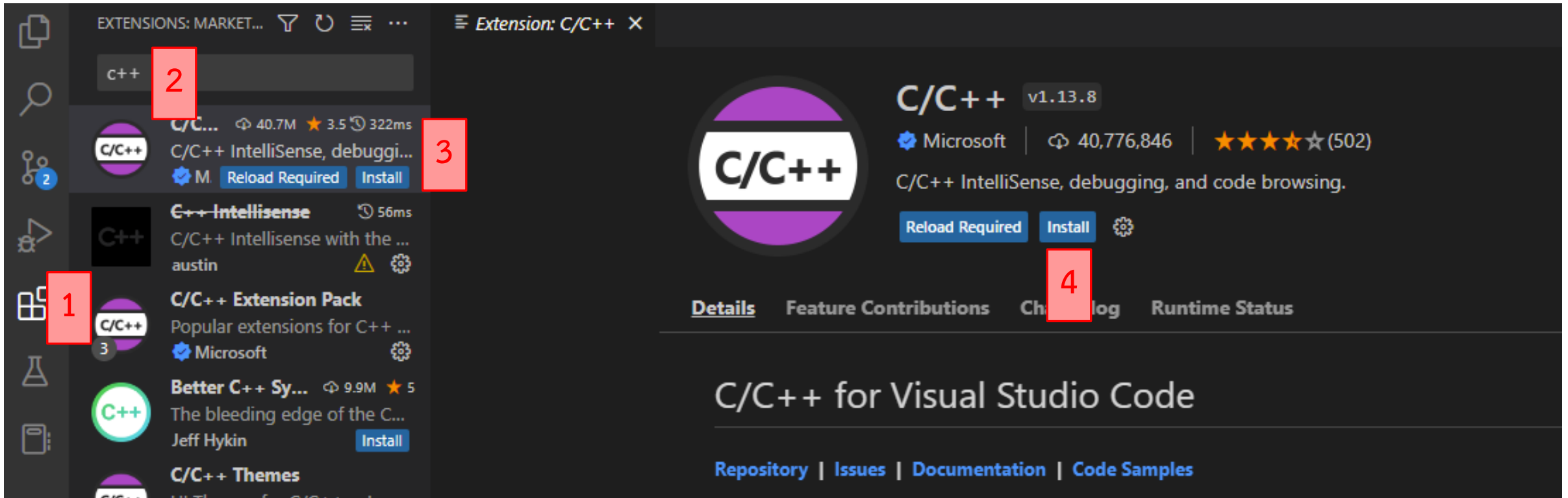
# Objective

---

- Install C++ compiler (VS CODE) ([Code block](#))
  - [Windows](#)
  - [MAC](#)
  - [Linux](#)
- Remind C programming skill



# VS code



1. click extension menu

2. Type c++

3. Find c/c++ IntelliSense, debugging and code browsing

4. Click install and Done!

## cout (standard stream out)

---

ส่งออกข้อมูลทาง standard output (เหมือน printf)

```
cout << a << "string" << b << c;
```

- ใช้ได้กับตัวแปรหลายชนิด (int float string)
- Chain call ได้ จะเริ่มส่งออกจากด้านซ้ายสุด
- ถ้าต้องการเว้นบรรทัดให้ chain endl ออกไป (end line)

```
string a = "haruhi";  
int i = 15532;  
float f = 1.414;  
  
cout << "hello" << endl;  
cout << a << " " << i;  
cout << f << endl;
```

```
hello  
haruhi 155321.414
```

# C++ function

---

- function declaration

- ประกาศ function กำหนด ชื่อ, return type, parameter
- สามารถใส่ definition ลงไปได้

- function definition

- ส่วนการทำงานของ function
- จับคู่กับ declaration ที่เคยประกาศมาแล้ว



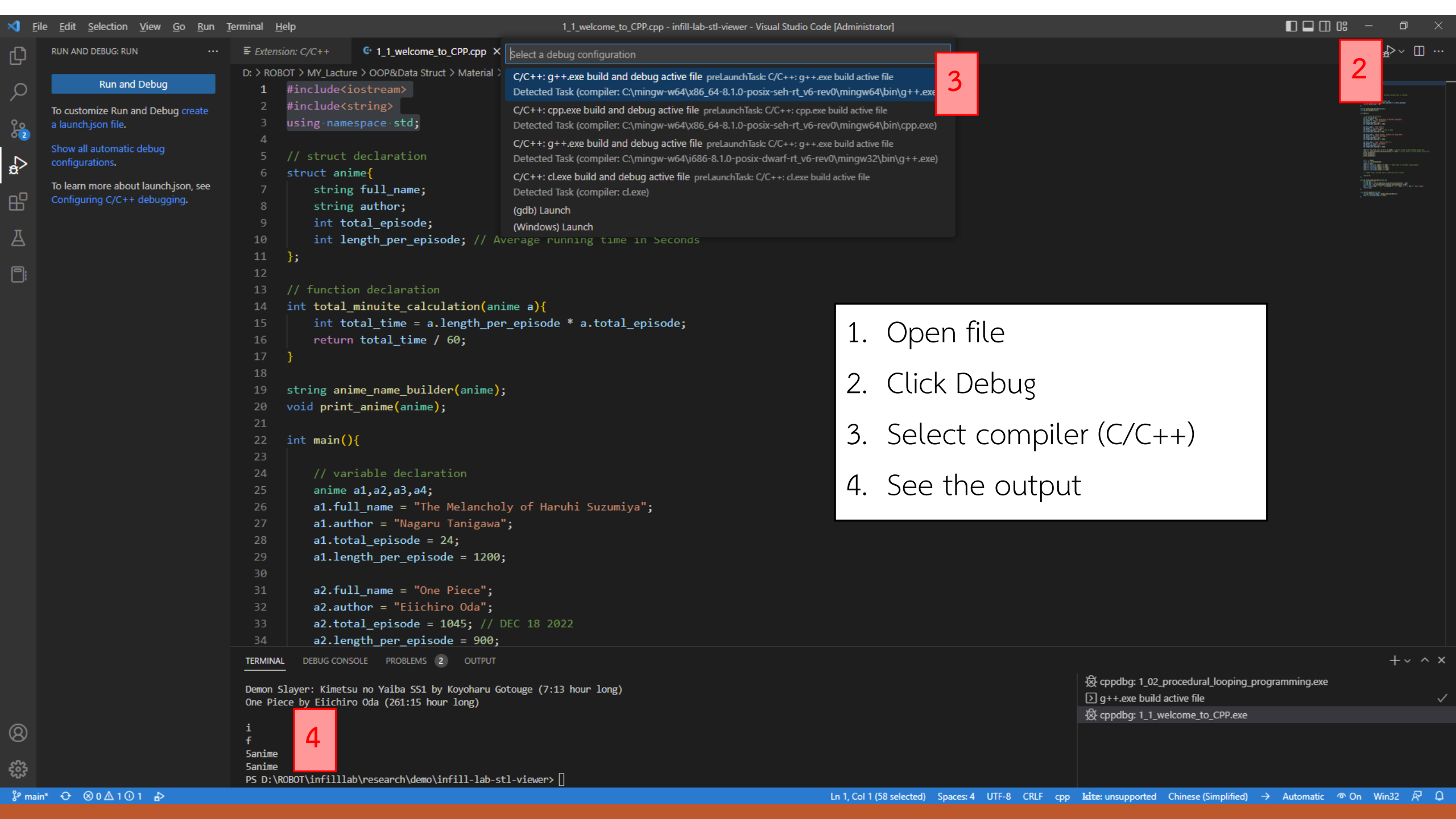
## Declaration

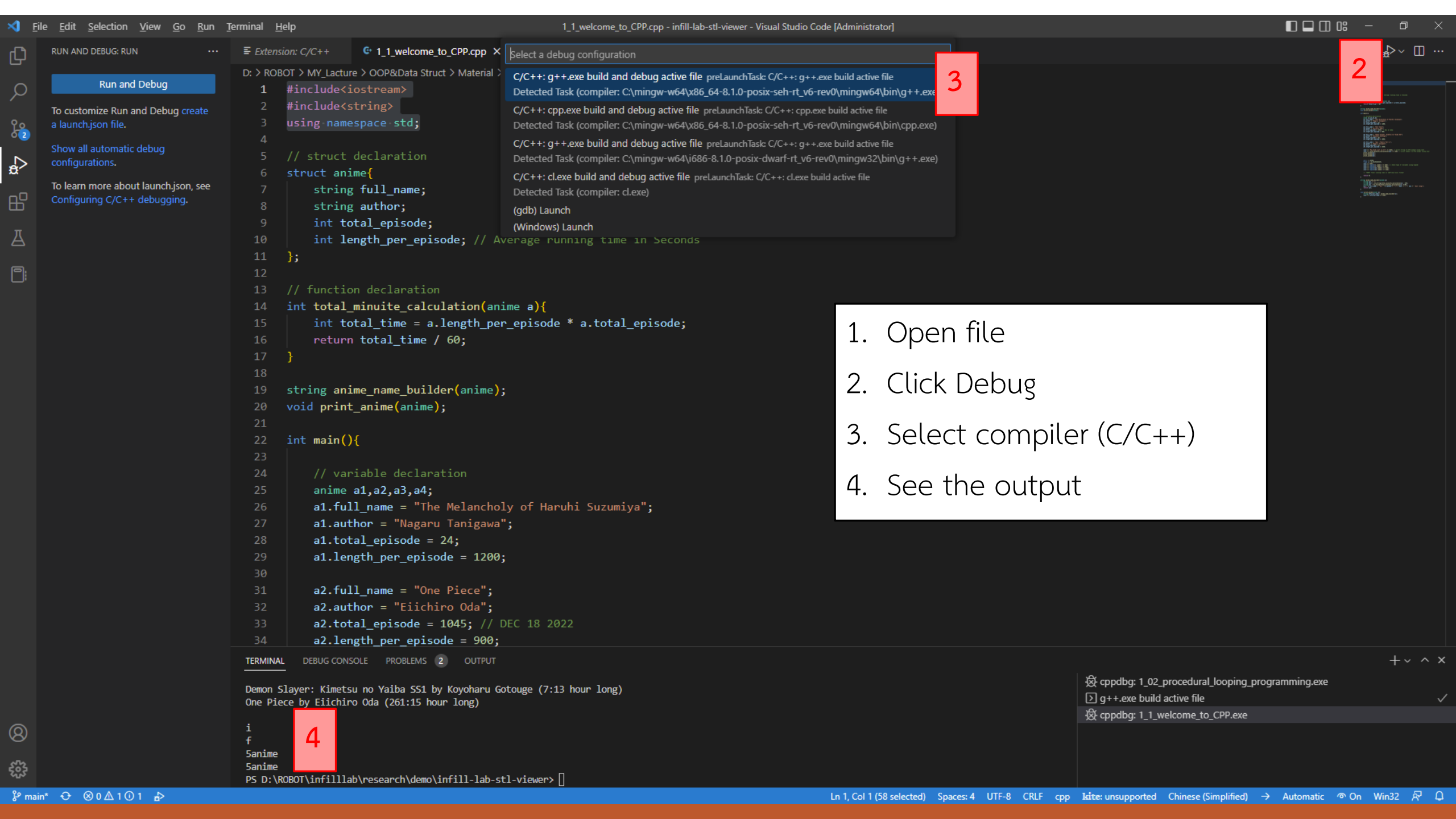
```
12
13 // function declaration
14 int total_minuite_calculation(anime a){
15     int total_time = a.length_per_episode * a.total_episode;
16     return total_time / 60;
17 }
18
19 string anime_name_builder(anime);
20 void print_anime(anime);
21
```

## Main

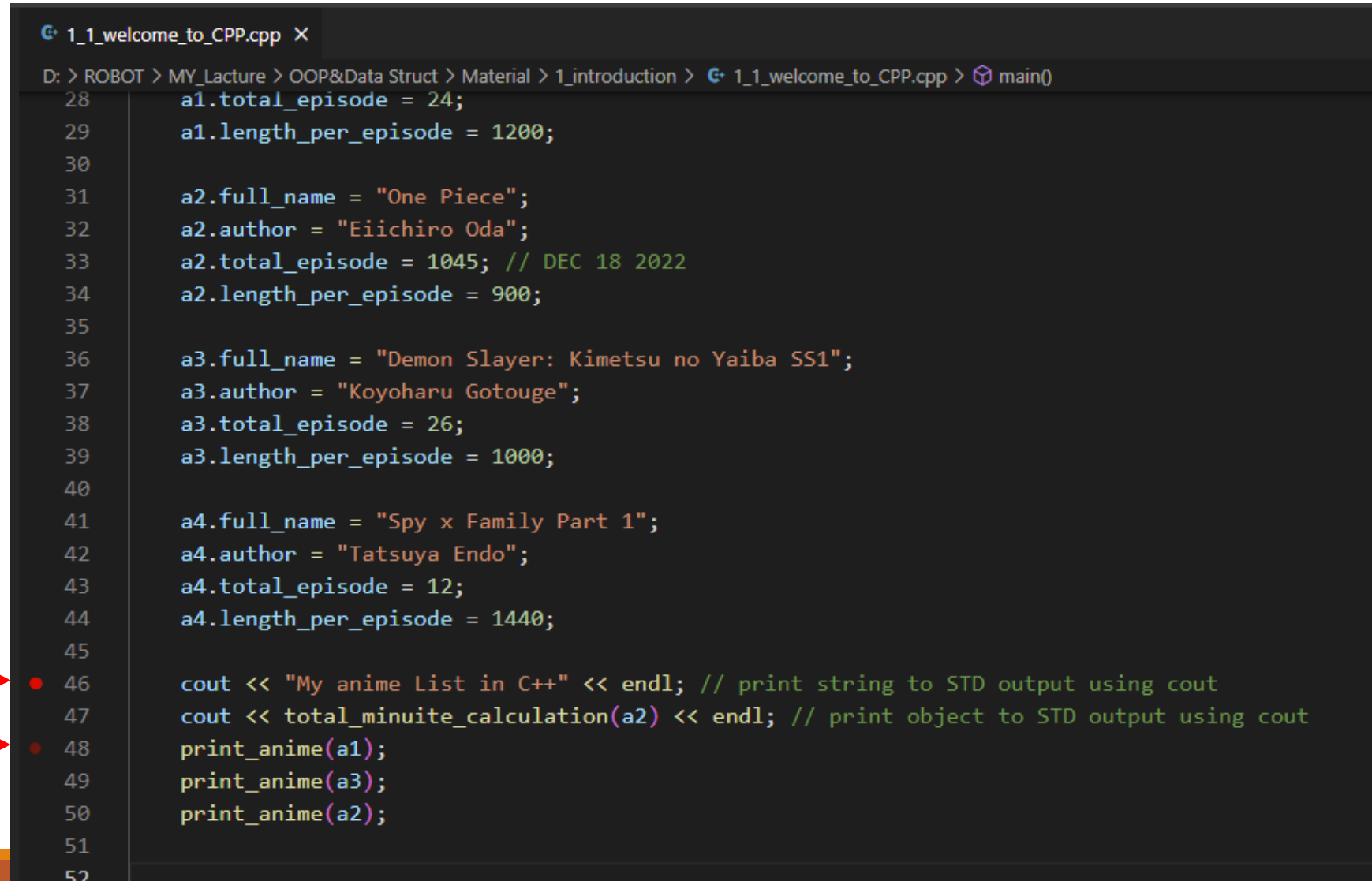
## Definition

```
66 // function definition
67 string anime_name_builder(anime a){
68     string out = "";
69     string hour = to_string(total_minuite_calculation(a) / 60);
70     string min = to_string(total_minuite_calculation(a) % 60);
71     out = a.full_name + " by " + a.author + " (" + hour + ":" + min + " hour long)";
72     return out;
73 }
74
75 void print_anime(anime a){
76     string official_name = anime_name_builder(a);
77     cout << official_name << endl;
78 }
```





# Breakpoint



The image shows a code editor window titled "1\_1\_welcome\_to\_CPP.cpp" with a dark theme. The code defines four anime objects (a1, a2, a3, a4) and prints their details. A debugger interface is visible at the bottom, showing two breakpoints (red dots) on lines 46 and 48. To the left of the editor, two white boxes labeled "Breakpoint" have red arrows pointing to these lines.

```
28     a1.total_episode = 24;
29     a1.length_per_episode = 1200;
30
31     a2.full_name = "One Piece";
32     a2.author = "Eiichiro Oda";
33     a2.total_episode = 1045; // DEC 18 2022
34     a2.length_per_episode = 900;
35
36     a3.full_name = "Demon Slayer: Kimetsu no Yaiba SS1";
37     a3.author = "Koyoharu Gotouge";
38     a3.total_episode = 26;
39     a3.length_per_episode = 1000;
40
41     a4.full_name = "Spy x Family Part 1";
42     a4.author = "Tatsuya Endo";
43     a4.total_episode = 12;
44     a4.length_per_episode = 1440;
45
46     cout << "My anime List in C++" << endl; // print string to STD output using cout
47     cout << total_minuite_calculation(a2) << endl; // print object to STD output using cout
48     print_anime(a1);
49     print_anime(a3);
50     print_anime(a2);
51
52
```

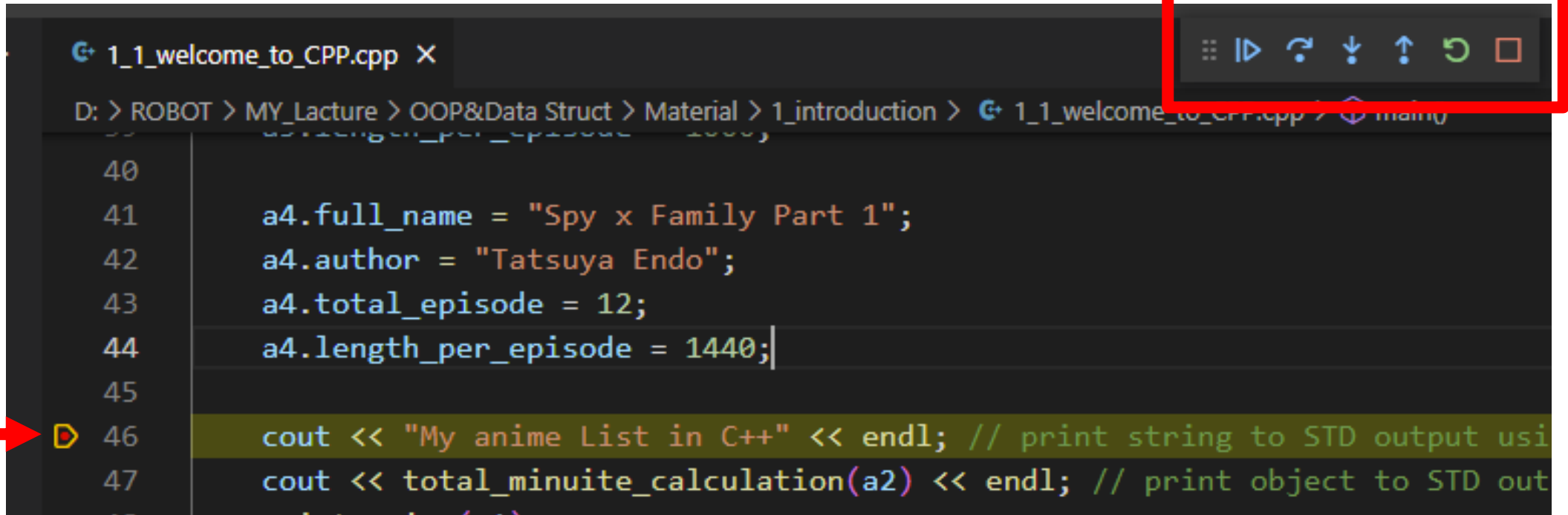
Breakpoint

Breakpoint

# Debugging mode

ให้โปรแกรมทำงานทีละบรรทัด (line of code)

Debugging tools

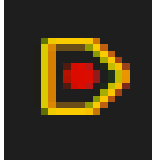


The screenshot shows a code editor with a file named `1_1_welcome_to_CPP.cpp`. The code is written in C++ and includes several lines of initialization and output. The line indicator on the left shows lines 40 through 47. Line 46 is highlighted in green, and a yellow bug icon is next to it. The debugging toolbar on the right contains icons for running, stepping through, and other debugging actions. A red box highlights the debugging toolbar, and a red arrow points from the 'Debugging tools' label to it. Another red arrow points from the 'Line indicator' label to the line indicator.

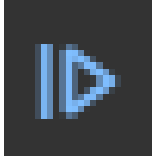
```
40
41     a4.full_name = "Spy x Family Part 1";
42     a4.author = "Tatsuya Endo";
43     a4.total_episode = 12;
44     a4.length_per_episode = 1440;
45
46     cout << "My anime List in C++" << endl; // print string to STD output usi
47     cout << total_minuite_calculation(a2) << endl; // print object to STD out
```

Line indicator

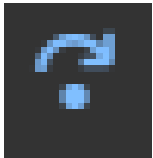
# Debugging mode



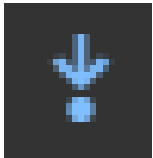
Line indicator - ตอนนี้โปรแกรมรันถึงบรรทัดที่เท่าไร



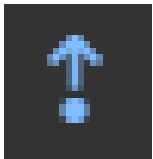
Continue – รันจนถึง breakpoint ต่อไป



Step Over – รันทีละบรรทัด (ไม่เข้าไปใน stack call หรือ function)



Step Into – รันทีละคำสั่งรวมไปถึง stack call (เข้าไปใน Function)



Step Out – รันจนกระทั่งออกจาก stack call หรือ function



Restart – เริ่ม Debug ใหม่

# Lab 001

---

"SHOW" Total running time in (DAY:hour:min) format

ให้แสดงเวลาฉายรวมของทุกเรื่องในรูปแบบ (DAY:hour:min)