C++ Basics

Lab 10

TA : Hyuna Seo, Kichang Yang, Minkyung Jeong, Jae Yong Kim

SEOUL NATIONAL UNIVERSITY

Announcement

- You should finish the lab practice and submit your job to eTL before the next lab class starts(Wednesday, 7:00 PM).
- The answer of the practice will be uploaded after the due.

Goal of this Lab

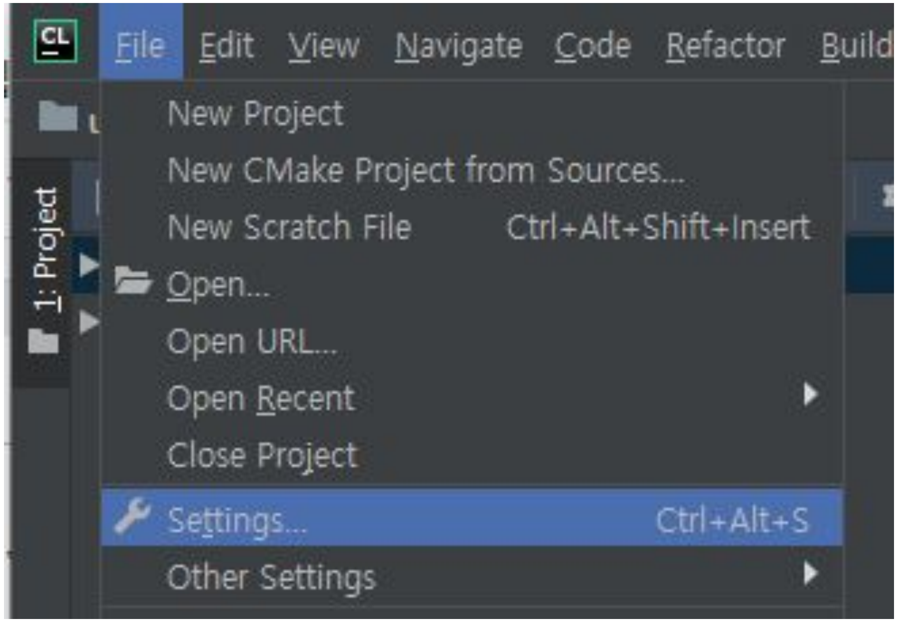
- Understand how to compile C++ program with multiple source files.
- Overview and exercise the basic C++ syntax.

Overview

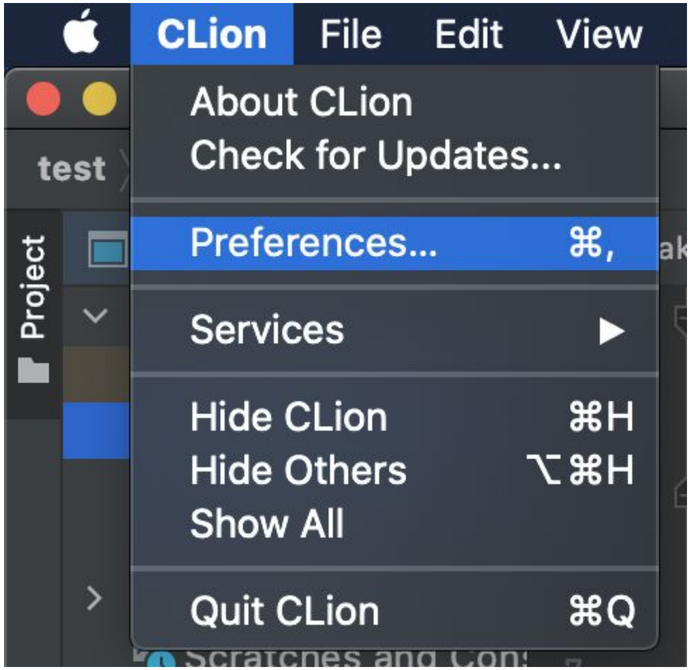
- **Build the program with multiple source files**
- Exercise basics of C++

Building multiple single-source files

- [Windows] File -> Settings
- [Mac] CLion -> Preferences



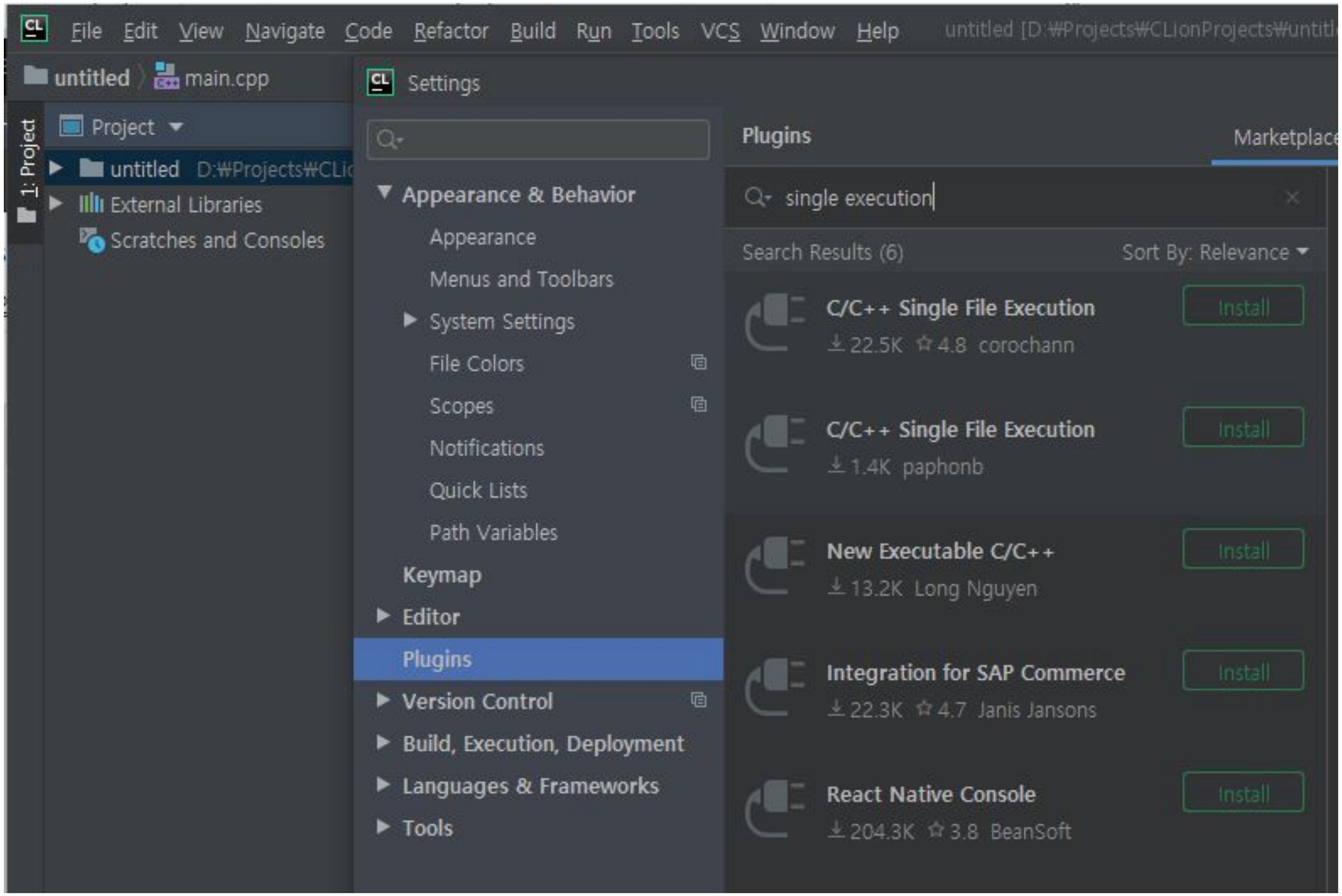
Windows



Mac

Building multiple single-source files

- Plugins -> search “single execution”



single execution

Search Results (6)

C/C++ Single File Execution
22.5K ☆ 4.8 corochann

C/C++ Single File Execution
1.4K paphonb

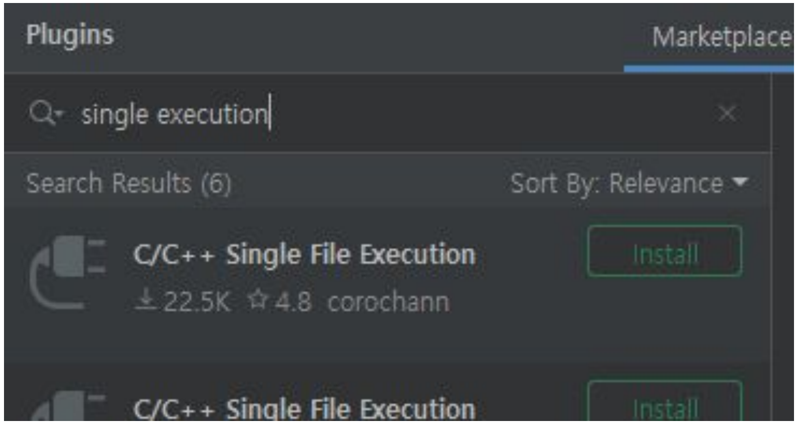
New Executable C/C++
13.2K Long Nguyen

Integration for SAP Commerce
22.3K ☆ 4.7 Janis Jansons

React Native Console
204.3K ☆ 3.8 BeanSoft

Building multiple single-source files

- Install “C/C++ Single File Execution”
  a. Select one with the tag “corochann”

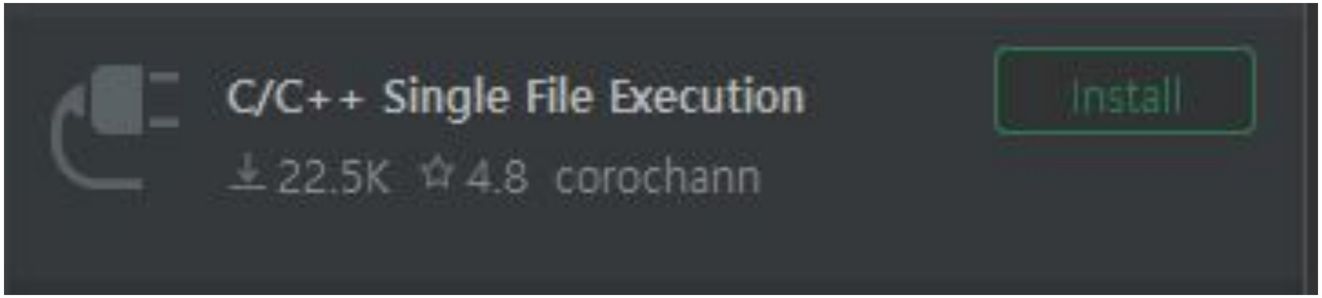


single execution

Search Results (6)

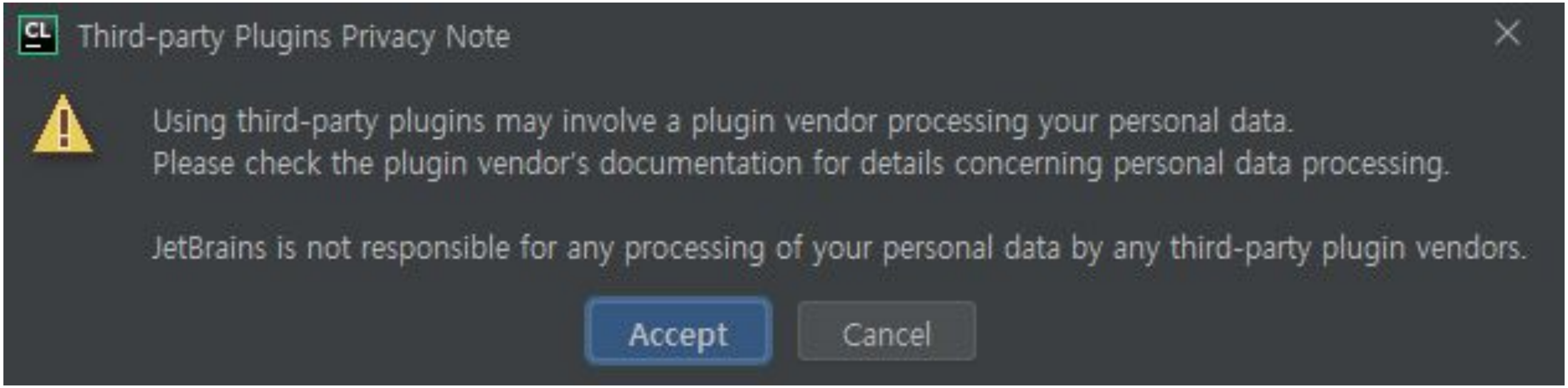
C/C++ Single File Execution
22.5K ☆4.8 corochann

C/C++ Single File Execution


C/C++ Single File Execution
22.5K ☆4.8 corochann

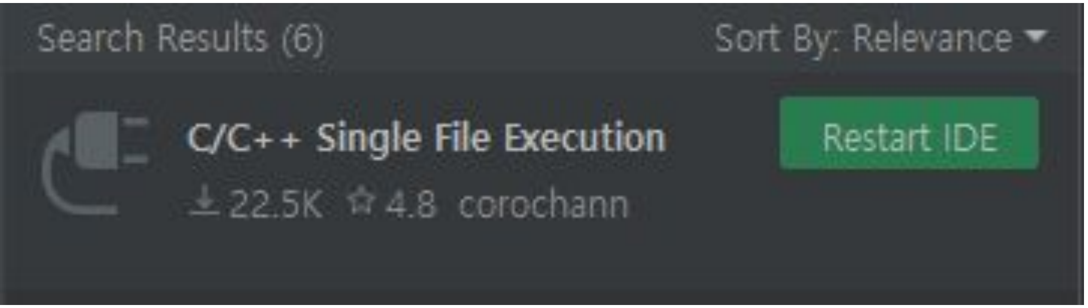
Building multiple single-source files

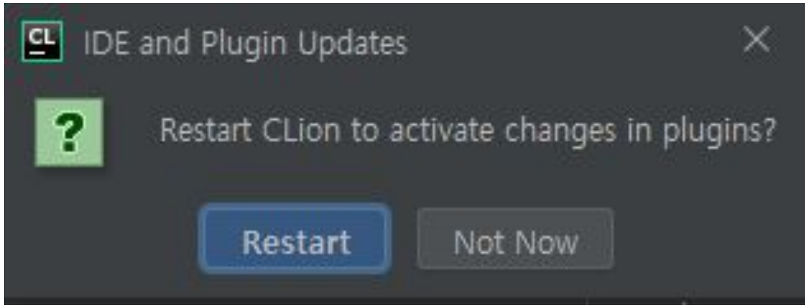
- Accept the third-party plugin privacy Note
- Click “Restart IDE”



Using third-party plugins may involve a plugin vendor processing your personal data.
Please check the plugin vendor's documentation for details concerning personal data processing.

JetBrains is not responsible for any processing of your personal data by any third-party plugin vendors.

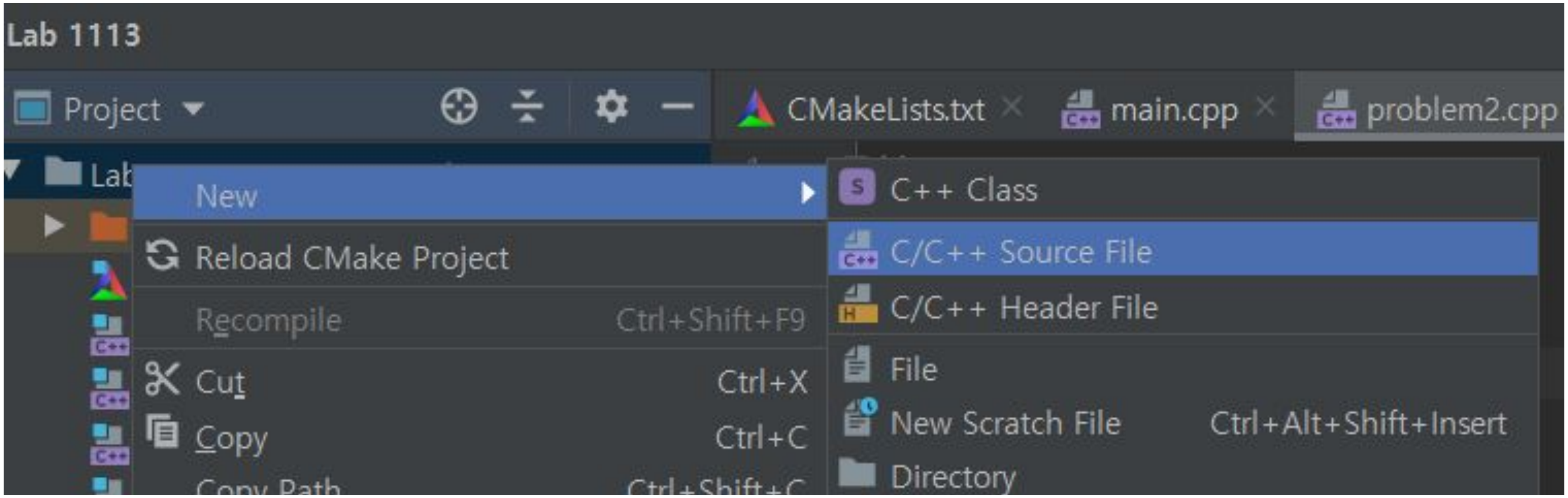




Restart CLion to activate changes in plugins?

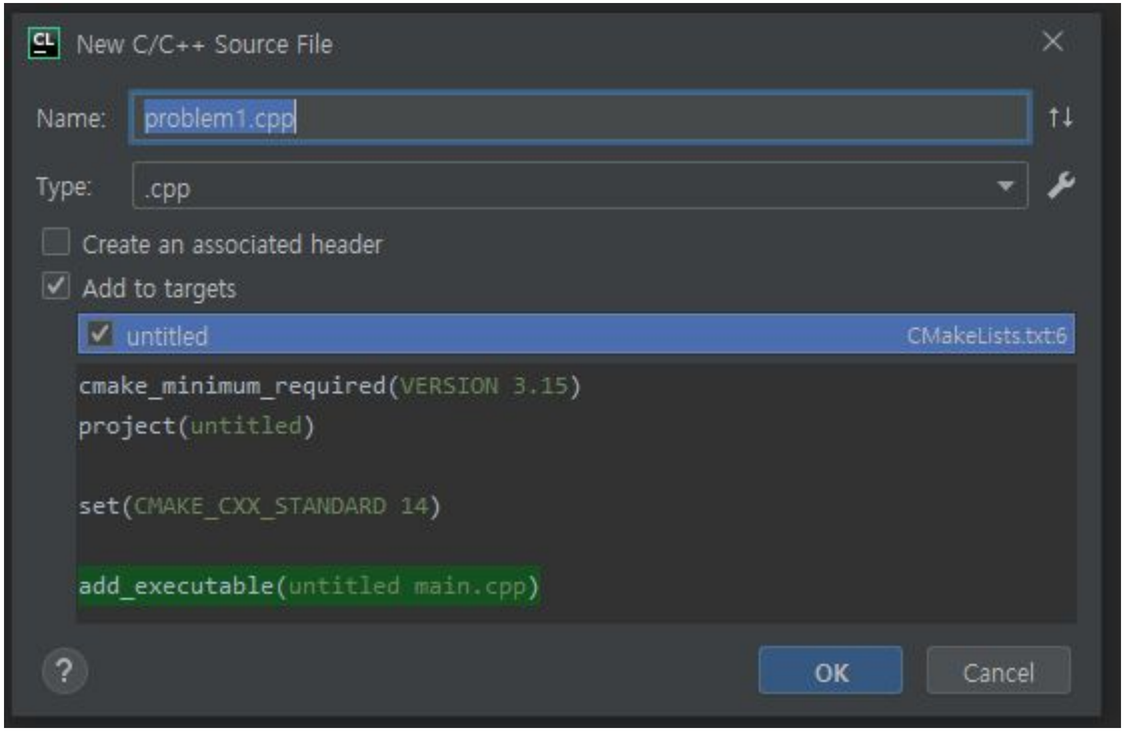
Building multiple single-source files

- Make a new source file



Building multiple single-source files

- Make a new source file



Name: problem1.cpp

Type: .cpp

☐ Create an associated header

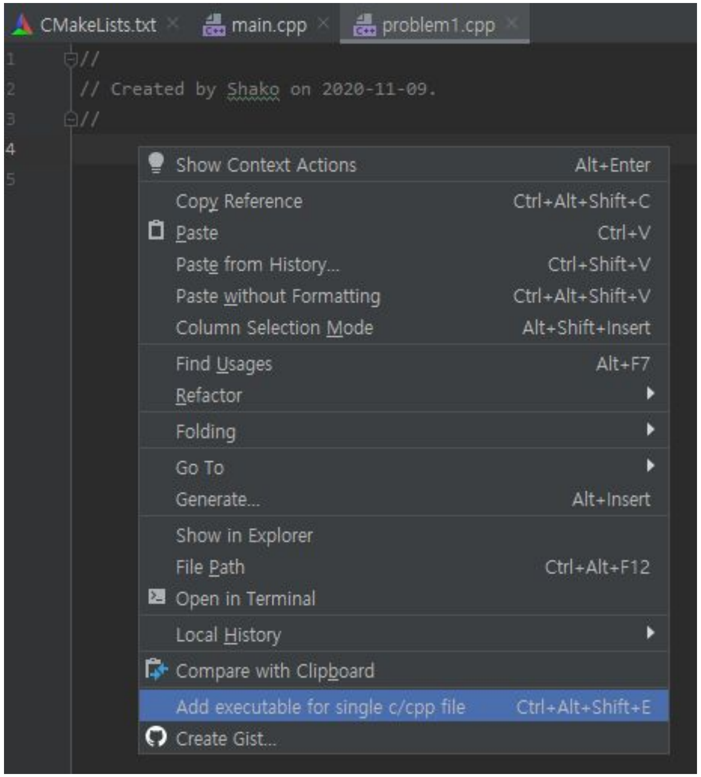
☑ Add to targets

☑ untitled CMakeLists.txt:6

```
cmake_minimum_required(VERSION 3.15)
project(untitled)

set(CMAKE_CXX_STANDARD 14)

add_executable(untitled main.cpp)
```

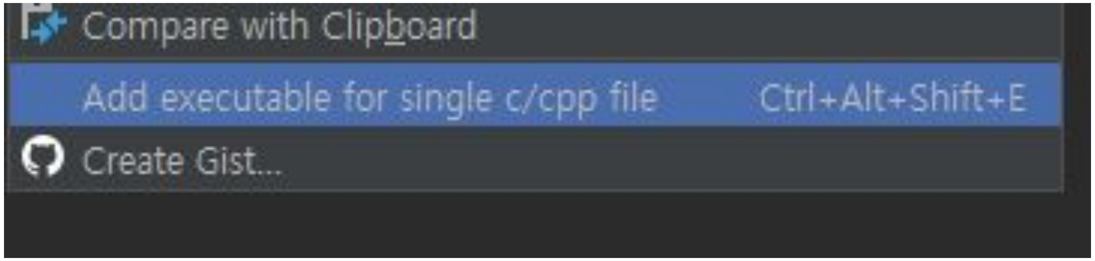
Building multiple single-source files

- Select a file to compile.
- Right click on the editor panel.
- Click “Add executable for single c/cpp file”



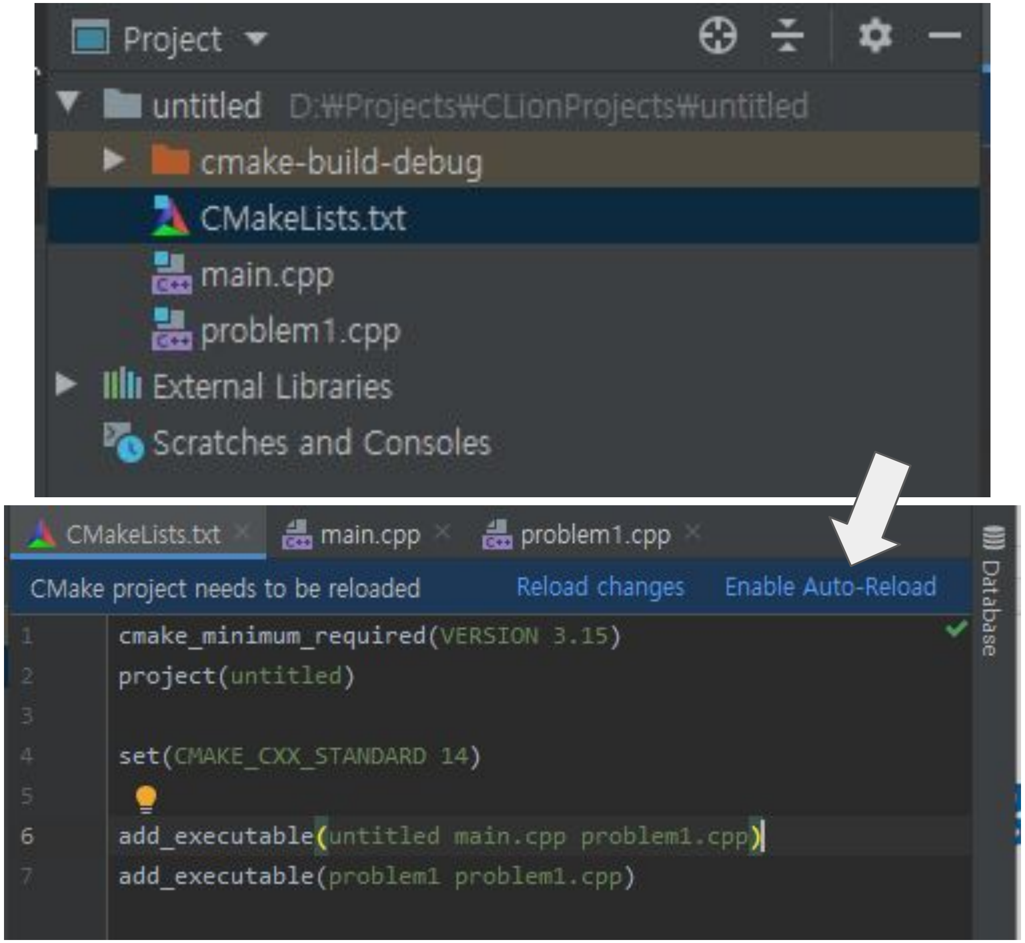
```
//
// Created by Shako on 2020-11-09.
//
```

CMakeLists.txt
main.cpp
problem1.cpp
Show Context Actions Alt+Enter
Copy Reference Ctrl+Alt+Shift+C
Paste Ctrl+V
Paste from History... Ctrl+Shift+V
Paste without Formatting Ctrl+Alt+Shift+V
Column Selection Mode Alt+Shift+Insert
Find Usages Alt+F7
Refactor
Folding
Go To
Generate... Alt+Insert
Show in Explorer
File Path Ctrl+Alt+F12
Open in Terminal
Local History
Compare with Clipboard
Add executable for single c/cpp file Ctrl+Alt+Shift+E
Create Gist...



Building multiple single-source files

- Go to CMakeLists.txt and Click Enable Auto-reload

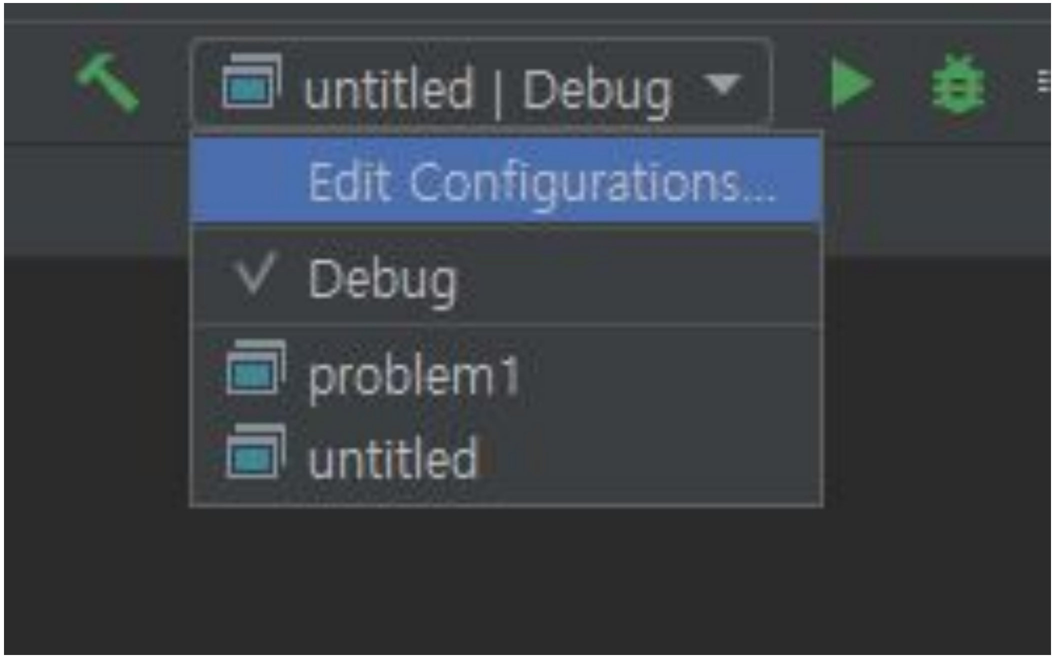


CMake project needs to be reloaded

```
cmake_minimum_required(VERSION 3.15)
project(untitled)

set(CMAKE_CXX_STANDARD 14)

add_executable(untitled main.cpp problem1.cpp)
add_executable(problem1 problem1.cpp)
```

Building multiple single-source files

- Choose the main execution target you want.



Overview

- C++ environment setting
- Build the program with multiple source files
- **Exercise basics of C++**
  - Lecture
  - Problem 1 ~ 6 (each 5 min)

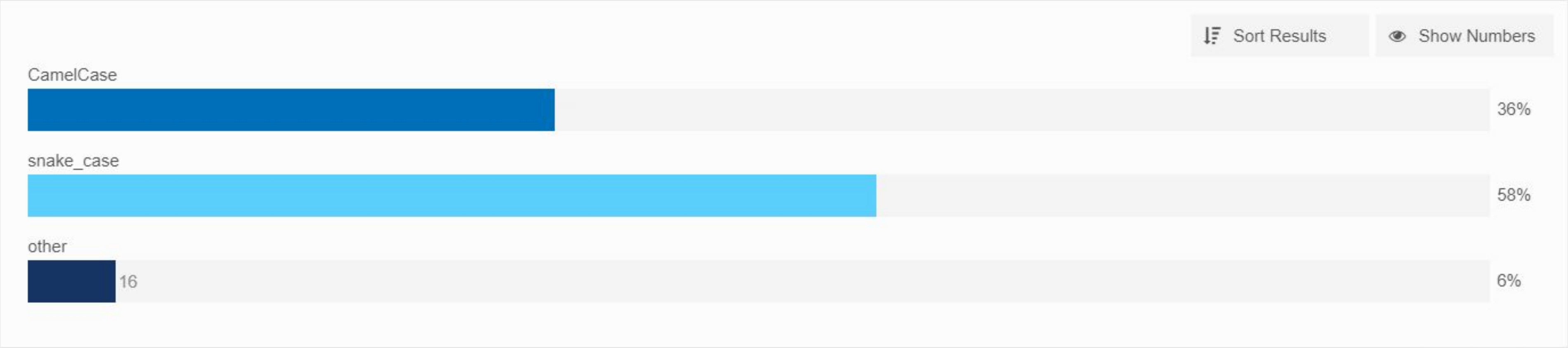
Java vs C++

- Headers / Namespace

- Characteristics of C
  - Pointers and References
  - Pre-processors (Macros)

Variable naming convention

- There is no strong standard of naming variables in C++
- Standard library uses snake_case for variables and methods
- Choose what you like, but be always consistent with your naming convention



Header (.h) File

- C++ libraries separate declarations and implementations for variables, functions, and classes.
- Declarations are in header (.h) files whereas implementations are in body (.cpp) files.
- Importing a header file allows to use corresponding implementations.
- Header file also prevents multiple inclusion of the same implementations.

Headers and Namespace

```
import java.util.Scanner  vs  #include <iostream>
```

- Make declarations in a header file, then use the #include directive in every .cpp file or other header file that requires that declaration. The #include directive inserts a copy of the header file directly into the .cpp file prior to compilation.

- Now what if several header files have the same exact function name and parameters?

Headers and Namespace

```
void printAll(){
    //print something
}
void printAll(){
    //print something
}
int main(void){
    printAll();
    return 0;
}
```

Headers and Namespace

```
namespace A{
   void printAll(){
      //print something
   }
}

namespace B{
   void printAll(){
      //print something
   }
}
```

```
int main(void){
   A::printAll();
   B::printAll();
   return 0;
}
```

1. Using the scope resolution operator “::”
2. Using the keyword “using”

Headers and Namespace

```
namespace A{
   void printAll(){
      //print something
   }
}

namespace B{
   void printAll(){
      //print something
   }
}
```

```
using namespace A;

int main(void){
   printAll();
   B::printAll();
   return 0;
}
```

1. Using the scope resolution operator “::”
2. Using the keyword “using”

Input & Output

Console

```
output  Put an integer
 input  4
output  The first input is 4
output  Put an integer and a character
 input  2 d
output  The second input is 2, d
```

```
#include <iostream>

int main() {
  int var, ivar;
  char cvar;
  std::cout << "Put an integer" << std::endl;
  std::cin >> var;
  std::cout << "The first input is " << var << std::endl;
  std::cout << "Put an integer and a character"
            << std::endl;
  std::cin >> ivar >> cvar;
  std::cout << "The second input is " << ivar
            << ", " << cvar << std::endl;
}
```

String

- Check string equality with `==` operator. (Different from Java string comparison)

```
#include <iostream>

int main() {
    std::string str1 = "abcde",
                str2 = "abcde";
    bool is_equal = (str1 == str2);
    std::cout << is_equal << std::endl;
}
```

Output

```
1          # Meaning true
```

Global Variables and Functions

```
#include <iostream>

int glob = 123; // Global variable declaration

int func(int i) { // Global function declaration
    return glob + i;
}

int main () {
    int local = 111; // Local variable declaration
    std::cout << func(local) << std::endl;
}
```

Output

```
234
```

Arrays Declaration

- Like Java, arrays are used to store multiple values in a single variable.
- Declare an array with the variable type, the name of the array followed by square brackets, and specify the number of elements to store.
    - It is different from Java array declaration.

```
#include <string>
int iarr[5];
string sarr[5];
```

Array Initialization

- Use array literal to declare an array with initialization.
- Place the values in a comma-separated list inside curly braces.
- The size of the array can be omitted.

```
#include <string>

int nums1[3] = {10, 20, 30},
    nums2[] = {10, 20, 30};
string cars1[4] = {"Volvo", "BMW", "Ford", "Mazda"},
    cars2[] = {"Volvo", "BMW", "Ford", "Mazda"};
```

Access an Array Element

- Access/change an array element by referring to the index number.

```
#include <iostream>
#include <string>
using namespace std;

int main () {
    string cars[] = {"Volvo", "BMW", "Ford", "Mazda"};
    cout << cars[0] << endl;
    // This statement changes the value of the first element in cars
    cars[0] = "Opel";
    cout << cars[0] << endl;
}
```

Output

```
Volvo
Opel
```

Loop Through an Array

- Loop through array elements with a loop.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
  string cars[] = {"Volvo", "BMW", "Ford", "Mazda"};
  // Outputs all elements in the cars array:
  for (string car : cars) { cout << car << endl; }
}
```

Output

```
Volvo
BMW
Ford
Mazda
```

Conditional statement

- Use if, else, and else if to specify a block of code to execute depending on a condition.

```
#include <iostream>
using namespace std;

int main () {
    int time = 22;
    if (time < 10) {
       cout << "Good morning." << endl;
    } else if (time < 20) {
       cout << "Good day." << endl;
    } else {
       cout << "Good evening." << endl;
    }
 }
```

Output

```
Good evening.
```

Loop

```
#include <iostream>
using namespace std;

int main() {
  int i = 0;
  while (i < 5) {
    cout << i++ << endl;
  }

  i = 0;
  do {
    cout << i++ << endl;
  }
  while (i < 5);
}
```

Output

```
0
1
2
3
4
0
1
2
3
4
```

Loop

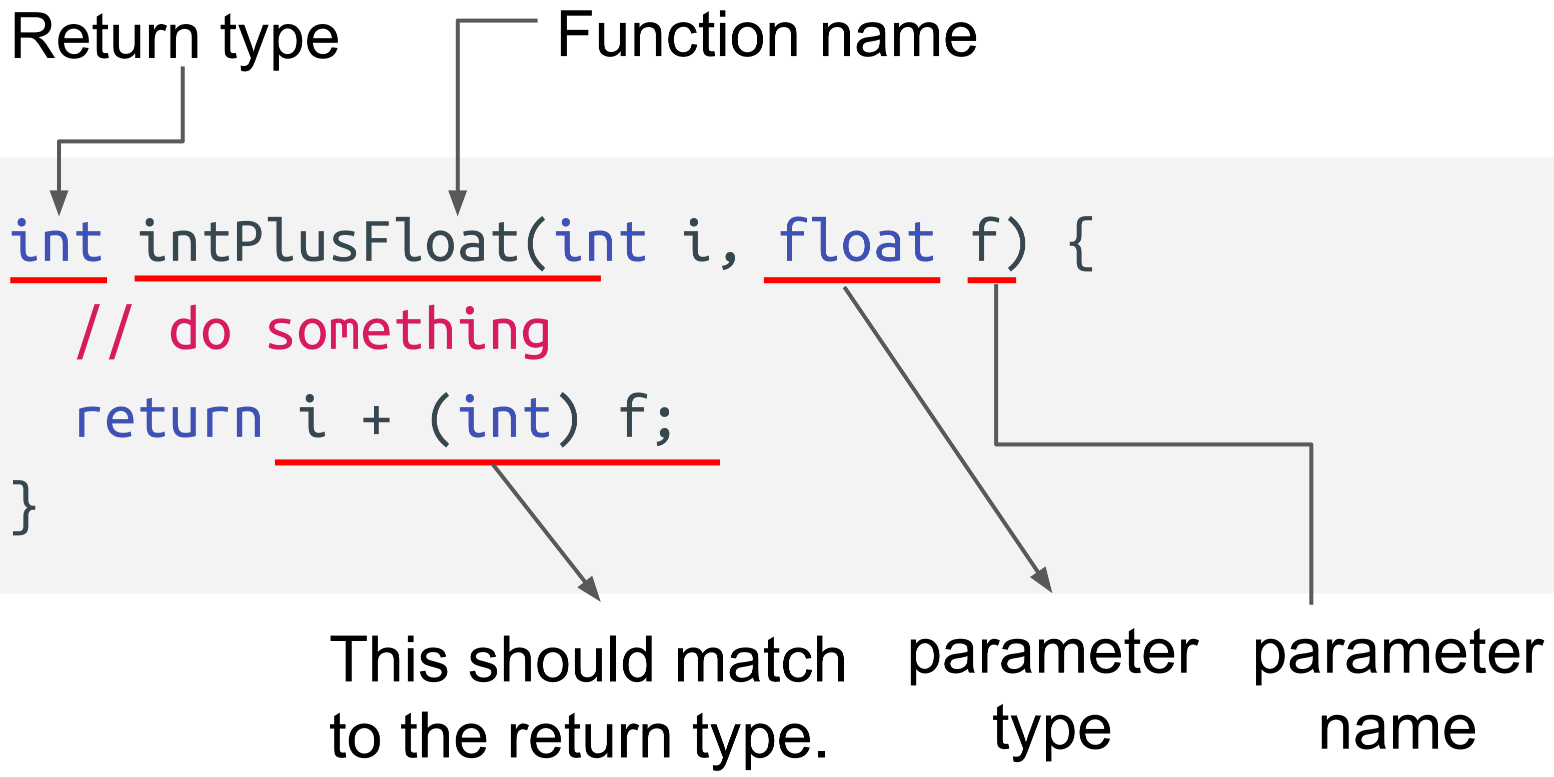
```cpp
#include <iostream>
#include <iterator>
using namespace std;

int main() {
   int arr[] = {0, 1, 2, 3, 4};

   for (int i = 0; i < size(arr); i++) {
     cout << arr[i] << endl;
   }

   for (int i : arr) {
     cout << i << endl;
   }
}
```

Output

```
0
1
2
3
4
0
1
2
3
4
```

Function

- A function is a block of code which runs when it is called.



Return type

Function name

```
int intPlusFloat(int i, float f) {
  // do something
  return i + (int) f;
}
```

This should match to the return type.

parameter type

parameter name

Function Overloading

- Multiple functions can have the same name with different parameters and return types.

```cpp
#include <iostream>
using namespace std;

int add(int x, int y) { return x + y; }
double add(double x, double y) { return x + y; }

int main() {
  cout << add(1, 2) << endl;
  cout << add(1.2, 3.4) << endl;
}
```

Output

```
3
4.6
```

Macro

```
#include <iostream>
using namespace std;

#define PI 3.14

int main() {
    double radius = 10;
    double circumference = 2 * PI * radius;
    cout << circumference << endl;
}
```

Output

```
62.8
```

Macro

```
#include <iostream>
using namespace std;

#define SUB(x,y) x-y
#define PRINT(x) cout << x << endl;

int main() {
    int k = 10;
    int m = 5;
    int diff = SUB(k,m);
    PRINT(diff);
}
```

Output

```
5
```

Write files

- To create a file, use either `ofstream` or `fstream` object, and specify the name of the file.
- Use the insertion operator `<<` to write to the file.
- Close the stream object when the writing is done.

```cpp
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  // Create and open a text file
  ofstream my_file("filename.txt");
  // Write to the file
  my_file << "Files are fun!" << endl;
  my_file.close(); // Close the file
}
```

filename.txt

```
Files are fun!
```

Read files

```
#include <iostream>
#include <fstream>

using namespace std;
int main() {
  string str;
  ifstream my_file("filename.txt");
  while (getline(my_file, str)) {
    cout << str << endl;
  }
  my_file.close();
}
```

filename.txt

```
1st line
2nd line
3rd line
```

Output

```
1st line
2nd line
3rd line
```

Pointers

- A variable (`foo` in the previous slide) that stores the address of another variable is called a ***pointer.***
- Pointers are powerful features that differentiate C++ from other programming languages like Java, JavaScript, Python, etc.

Usage of Pointers

- Modify variables inside another function.
- Optimize for the memory usage
  - e.g.) free unused space right away
- Dynamically allocate large memory space in the heap
- Implement advanced data structure like a linked list or tree
- Handle overriding and dynamic binding for inherited classes

Address-of Operator &

- Get a memory address with address-of operator &.

```
#include <iostream>
using namespace std;

int main() {
  int var = 3;
  cout << var << endl;
  cout << &var << endl;
}
```

Output

```
3
0x7ffeeeee1d7fc   # This can be different at each run
```

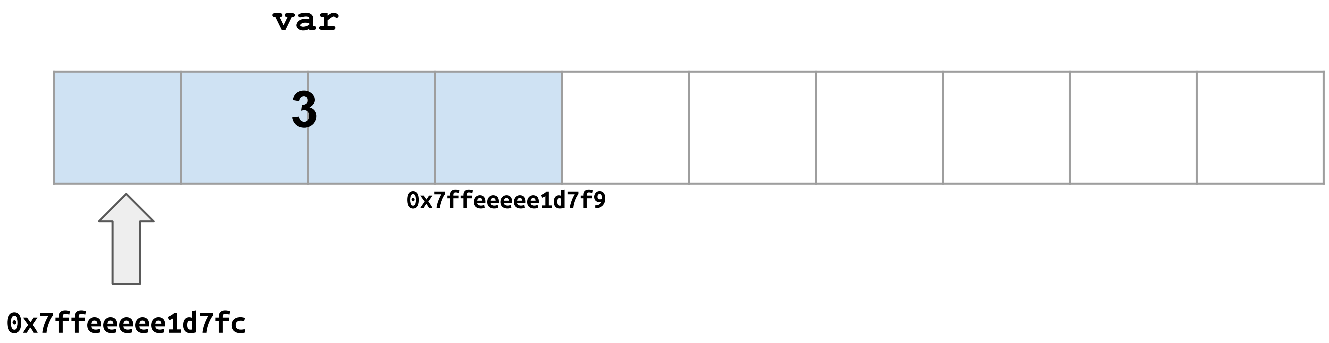
Pointers

- A pointer data type is created with `*` beside the existing data type. In the example below, `ptr` stores the address of an integer variable, `var`.

```cpp
#include <iostream>
using namespace std;
int main() {
  int var = 3;
  int* ptr = &var;
  cout << var << endl;
  cout << &var << endl;
  cout << ptr << endl;
}
```

Output

```
3
0x7ffeeaeb67fc
0x7ffeeaeb67fc
```

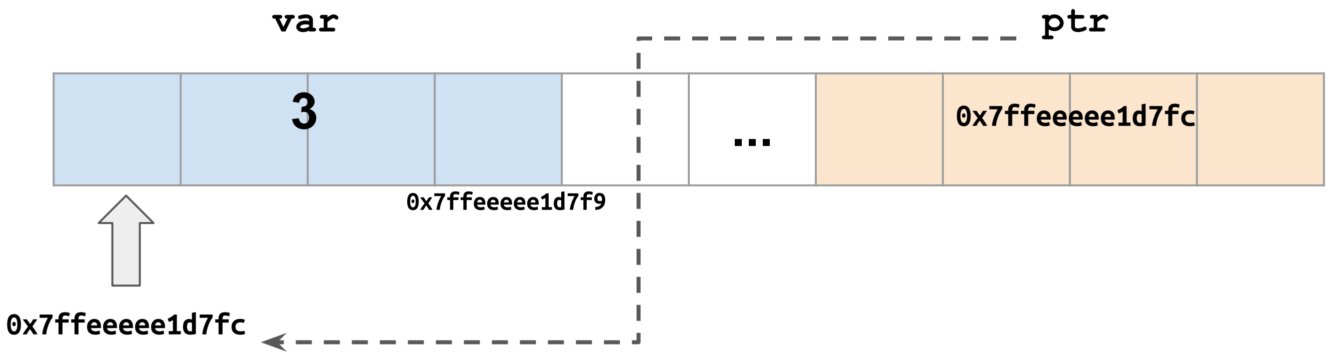


var

3

0x7ffeeeee1d7f9

0x7ffeeeee1d7fc



var

ptr

3

...

0x7ffeeeee1d7fc

0x7ffeeeee1d7f9

0x7ffeeeee1d7fc

Dereference Operator (*)

- You can access the value of a variable that a pointer points to, using the dereference operator *.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
  string food = "Pizza";
  string* ptr = &food;
  // Output the value of food
  cout << food << "\n";
  cout << ptr << "\n";
  // Access the memory address of ptr
     and output its value
  cout << *ptr << "\n";
}
```

Output

```
Pizza
0xba211ff940
Pizza
```

Pointer and Arrays

```
#include <iostream>
using namespace std;
int main() {
  int array[5] = {9, 7, 5, 3, 1};
  //print the value of the array variable
  std::cout << array << std::endl;

  //print address of the array elements
  std::cout << &array[0] << std::endl;
  return 0;
}
```

Output

```
0x77cddffd50

0x77cddffd50
```

- An array variable is actually a pointer that stores the address of the first element in the array!

Pointer and Arrays

```
#include <iostream>
using namespace std;
int main() {
  int array[5] = {9, 7, 5, 3, 1};
  int* ptr = array
  //dereference the array variable
  std::cout << *ptr << std::endl;

  //traverse array with pointer!
  std::cout << *(++ptr) << std::endl;
  return 0;
}
```

Output

```
0x77cddffd50

0x77cddffd50
```

Reference Variable

- A reference variable is a reference to an existing variable, and it is created with the & operator.

```
#include <iostream>
#include <string>
using namespace std;

int main() {
  string my_home = "My Home";
  string &my_house = my_home;
  cout << my_home << endl;
  cout << my_house << endl;
   my_house = “not anymore”;
  cout << my_home << endl;
}
```

Output

```
My Home
My Home
Not anymore
```

Swap with Reference Variables

- Easier swap with reference variables.

```cpp
#include <iostream>
using namespace std;
void swap(int& a, int& b) {
  int temp = a;
  a = b;
  b = temp;
}
int main() {
  int var1 = 1, var2 = 2;
  cout << var1 << ',' << var2 << endl;
  swap(var1, var2);
  cout << var1 << ',' << var2 << endl;
}
```

Output

```
1,2
2,1
```

Swap with Reference Variables

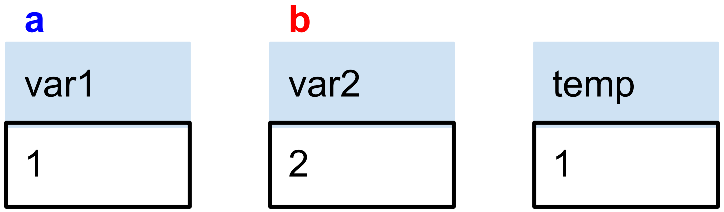
- Easier swap with reference variables.

```
#include <iostream>
using namespace std;
void swap(int& a, int& b) {
  int temp = a;
  a = b;
  b = temp;
}
int main() {
  int var1 = 1, var2 = 2;
  cout << var1 << ',' << var2 << endl;
  swap(var1, var2);
  cout << var1 << ',' << var2 << endl;
}
```

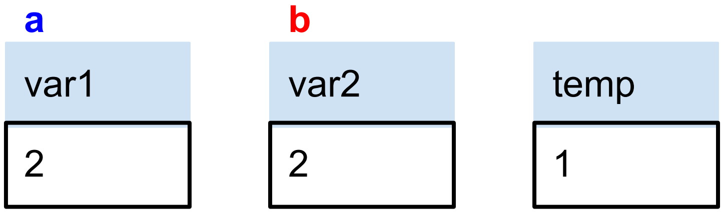
```
int& a = var1;
int& b = var2;
```

Output

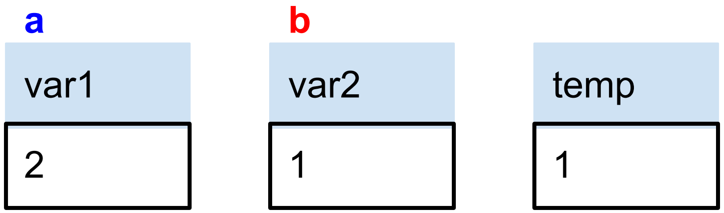
```
1,2
2,1
```



```
int temp = a; // temp = var1
```



```
a = b; // var1 = var2
```



```
b = temp; // var2 = temp
```

new & delete Keywords

- new keyword allocates a memory in the heap space, and return the pointer of the memory.
- delete keyword deletes the allocated memory which the pointer points.

```
#include <iostream>
using namespace std;

int main() {
    int *ptr = new int;
    cout << ptr << endl;
    delete ptr;
}
```

Output

```
0x7fbc6d4006a0
```

Smart Pointers

- In large programs with many programmers, it is hard to track all the pointers.
- Failing to handle pointers can lead to memory leak. Sometimes it causes fatal problems.

Smart Pointers

- C++ introduced **smart pointers** to avoid memory leak problems.
- Smart pointers are used to make sure that an object is deleted if it is no longer referenced. Programmers don’t have to care about deleting memories manually.
- There are three kinds of smart pointers; `unique_ptr`, `shared_ptr`, and `weak_ptr`
- You may get detailed information here: https://en.cppreference.com/book/intro/smart_pointers

Test Class

```
#include <iostream>

#include <memory>


using namespace std;

class Test{
Public:
    Test(int id){
        test_id = id;
        cout << “constructed” << endl;
    }

    ~Test(){
        cout << “destructed” << endl;
    }
Public:
    int test_id;

}
```

Unique Pointers

- A `unique_ptr` can be owned by only one owner.
- Cannot be copied or shared.

```cpp
#include <iostream>
#include <memory>


using std::unique_ptr; using std::make_unique;
int main() {
    unique_ptr<Test> test_unique1(new Test(1));
    unique_ptr<Test> test_unique2 = std::make_unique<Test>(2);
    //unique_ptr<test> test_unique3 = test_unique2; // this is not allowed
    std::cout << "id : " << test_unique1->test_id << std::endl;
    std::cout << "id : " << test_unique2->test_id << std::endl;
}
```

Output

```
constructed
constructed
id : 1
id : 2
destructed
destructed
```

Shared Pointers

- A `shared_ptr` can be owned by multiple owners.
- When no owner is using the object, it is destructed.
- Reference counting - deleted when reference count == 0

```
using std::shared_ptr; using std::make_shared;
shared_ptr<Test> test_shared() {
    shared_ptr<Test> test_shared1(new Test(1));
    shared_ptr<Test> test_shared2 = make_shared<Test>(2);
    shared_ptr<Test> test_shared3 = test_shared2;
    std::cout << "id : " << test_shared1->test_id << std::endl;
    std::cout << "id : " << test_shared2->test_id << std::endl;
    return test_shared3;
}
```

Shared Pointers (continued)

```
int main() {
    shared_ptr<Test> ptr = test_shared();
    std::cout << "id : " << ptr->test_id <<
std::endl;
    return 0;
}
```

Output

```
constructed
constructed
id : 1
id : 2
destructed
id : 2
destructed
```

Weak Pointers

- If two shared pointers point to each other, they are never released.
- `weak_ptr` pointing to a resource doesn’t affect the resource's reference count.
- When the last `shared_ptr` pointing the resource is destroyed, the resource will be freed, even if there are `weak_ptr` objects pointing to that resource.

```
int main() {
    shared_ptr<Test> test_shared1(new Test(1));
    shared_ptr<Test> test_shared2 = test_shared1;
    std::cout << "use count before : " << test_shared1.use_count() <<
std::endl;
    weak_ptr<Test> test_weak = test_shared1;
    std::cout << "id : " << test_weak.lock()->test_id << std::endl;
    std::cout << "use count after : " << test_weak.use_count() << std::endl;
    return 0;
}
```

Output

```
constructed
Use count
before: 2
Id : 1
Use count
before: 2
destructed
```

5 mins

Problem 1

Extend our hello world code using string comparison

- Input
  - ***name***, a single line of string from **stdin**
- Output
  - If the *name* is “Youngki”, print
    - “Hello, Professor!”
  - Otherwise, print
    - “Hello, (name)!”

5 mins

Problem 2

Write a code that calculates the area of a circle, using the following macros

#define PI 3.14159

#define AREA(r) ?????

- Input
  - *r*, a floating-point number
- Output
  - The area of a circle of radius *r*

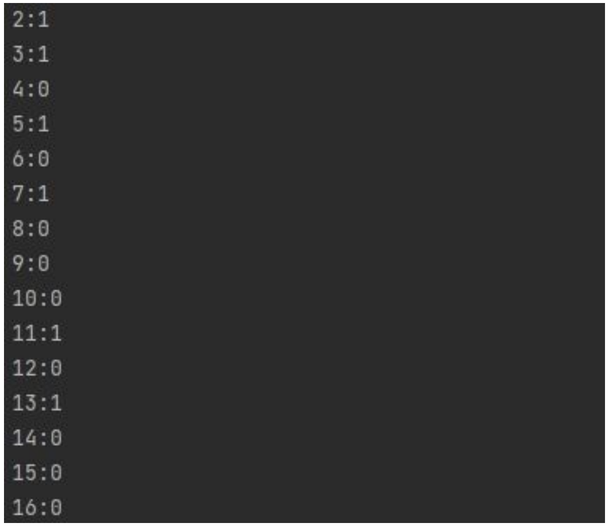
5 mins

Problem 3

Write a function that determines if a given natural number is prime or not. (You may write whatever you want in the main method.)

bool is_prime(int n)

- Input
  - ***n***, an integer, 2 <= n <= 100
- Output (return value)
  - true iff ***n*** is prime, false otherwise



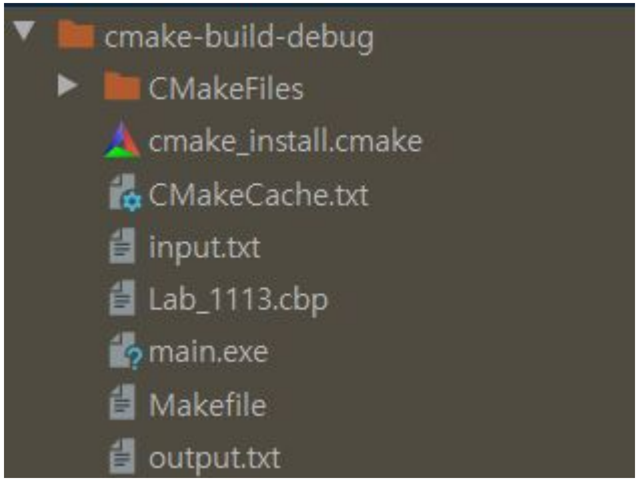
```
2:1
3:1
4:0
5:1
6:0
7:1
8:0
9:0
10:0
11:1
12:0
13:1
14:0
15:0
16:0
```

5 mins

Problem 4

Use the given file “input.txt” to get the parameter for function bool is_prime(int n) from problem 3 and write the output of the function to “output.txt”.

- Hint: change string to int using std::stoi(std::string)
- Place files in ***cmake-build-debug*** directory



```
▼ cmake-build-debug
    ▶ CMakeFiles
      cmake_install.cmake
      CMakeCache.txt
      input.txt
      Lab_1113.cbp
      main.exe
      Makefile
      output.txt
```

5 mins

Problem 5

Implement two 3-swap functions using both pointers and references. (You may write whatever you want in the main method.)

```
void three_swap(int *a, int *b, int *c);
```

```
void three_swap(int &a, int &b, int &c);
```

- Input
  - ***a, b, c***, 3 integers separated by whitespaces
- Output
  - ***a*** should be changed to ***b***, ***b*** to ***c***, and ***c*** to ***a***

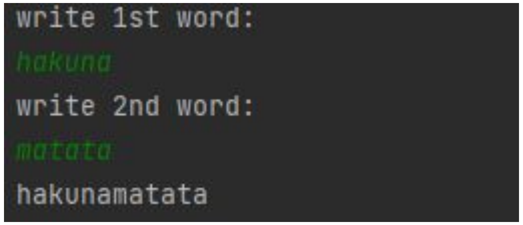
5 mins

Problem 6

Write a main program that receives 2 words from the user and concatenate those words through only using pointers and loops. Do not use ‘+’ or ‘strcat’ to complete the task.

Assume that the user does not write a string/word longer than 50 characters.

Hint: Strings are an array of characters!



```
write 1st word:
hakuna
write 2nd word:
matata
hakunamatata
```

Submission

- Compress the problem source files into a zip file.
  - It should include problem1.cpp ~ problem6.cpp
- Rename your zip file as 20XX-XXXXX_{name}.zip
  - for example, 2021-12345_YangKichang.zip
- Upload it to eTL - Lab 10 assignment.