Object-Oriented Programming: Moving from C to C++

Lectured by Ming-Te Chi 紀明德

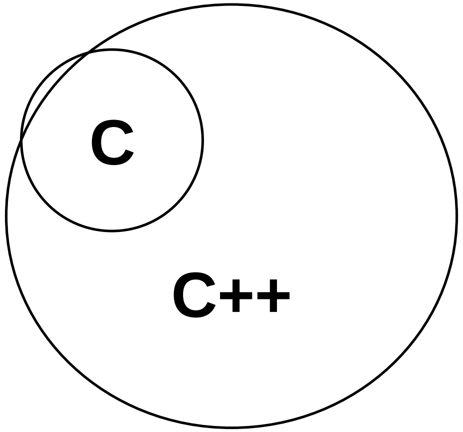
First Semester, 2022

Computer Science Department
National Chengchi University

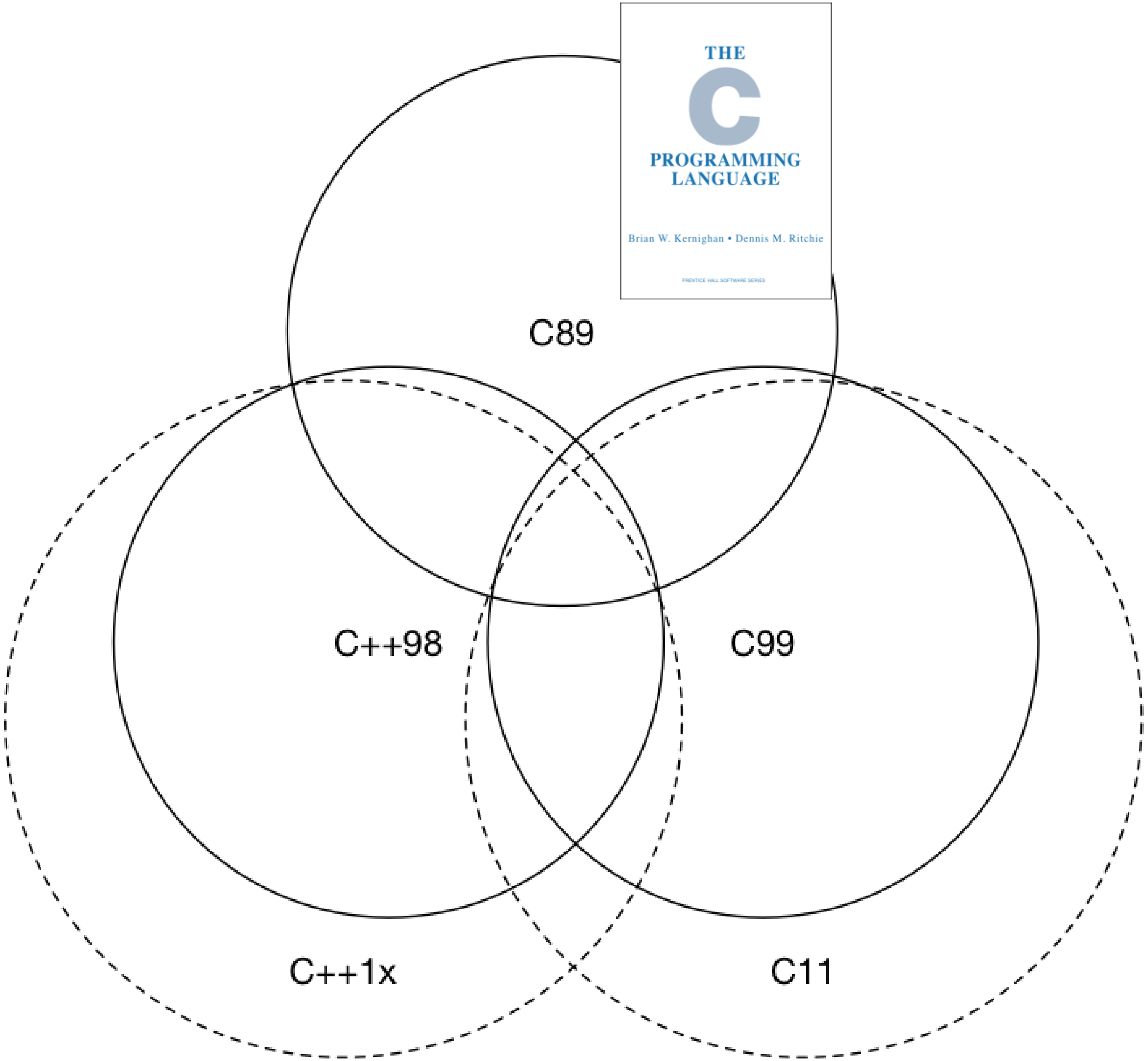
Slides credited from 李蔡彥 and 廖峻鋒

Why Expanding C to C++

- Providing a new approach to object-oriented programming
- Designing new programming tools that help write code more efficiently and more maintainable.
- Designing a more rigid programming language
- Creating a highly extendable programming language
- Enhancing some important C concepts such as functions, pointers, and structures



Modern C++ is not a superset of C


THE
C
PROGRAMMING
LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
PRENTICE HALL SOFTWARE SERIES
C89
C++98
C99
C++1x
C11

From Modern C++ Tutorial

C++ As A Better C
Differences Between C and C++

- Language grammar
  - New keywords
  - New comment format
  - Header files
  - Variable declaration
- Data types
  - `struct`, `enum`, and `union` type
  - Type conversion
  - Cast format
  - const storage type
- Better I/O (I/O stream)
- Namespaces

Keywords

- New keywords (part):

```
asm      inline
delete  new
class this  public  private protected friend virtual
template operator
try throw catch

nullptr      (C++11)
auto         (C++11)
constexpr    (C++11)
decltype     (C++11)
…
```

Comments

- End-of-line comments

C

```
if (b>a)
     return b; /* Could be also b>=
else
     return a; /* return 'a' if tie */
```

C++

```
if (b>a)
     return b; // Could be also b>=
else
     return a; // return 'a' if tie
```

Conclusion:

/* for multi-line comments */

// for single-line comments

First C++ Program [code]

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

int main() {
  // This is a comment.
  cout << "Hello, World! I am "
       << 8 << " Today!" << endl;
  return 0; // optional
}
```

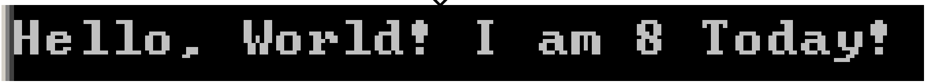
Include header file

Namespace

Print to screen

New line

Need to write **std::cout** if not using namespace



```
Hello, World! I am 8 Today!
```

Header Files

- Standard library header files: <>
  - Search path:
    system default: /usr/include
    specified: g++ -I/usr/local/include
- User-defined header files: ″ ″
  - Search path: current directory + standard path

```
#include <stdio.h>// C’s standard header
#include <iostream> // C++’s standard header
#include <iostream.h> // old style C++ header
#include “myheader.h” // user-specified header
```

Declare *vs*. Definition

- Declaration (宣告)
  - What: declaring parameters, return type, and name of a function
  - Usually in .h header files

```
int add(int a, int b);
```

or

```
int add(int, int);
```

- Definition (定義)
  - How: defining implementation details of a function

```
int add(int a, int b) {
    return a+b;
}
```

Variable Declaration

- Declaration location:
  - C: at the top of the function (before C99)
  - C++: anywhere in the function
- Example:

```
float ArrayAverage(int *array, int size) {
   int sum=0;
   for(int i=0;i<size;i++)
      sum += array[i];
   float avg;
   avg = (float)sum/(float)size;
   return avg;
}
```

- Reason: better readability (not necessarily)
- Scope: from the point of declaration to the end of the function

New bool Type

- bool: Boolean values (true or false)
  - Preferred to int
  - Taking less memory
  - Logically clearer

```
bool fun(float x, float y) {
   bool answer;
   if (x<y && x>0)
      answer = true;
   else
      answer = false;
   return answer;
}
```

struct, enum, and union Types

- No typedef's are needed any more.

C

```
typedef struct _point {
    float x, y;
} point;
point p; //struct _point p
```

C++

```
struct point {
    float x, y;
};
point p;
```

Anonymous union:

You can define a union without a tag or name.

```
struct iorf {
    int which_one;
    union {
        int i;
        float f;
    };
};
```

You can access the anonymous fields directly.

```
iorf uncertain;
uncertain.i = 1;
uncertain.f = 1.0f;
```

Type Conversion (I)

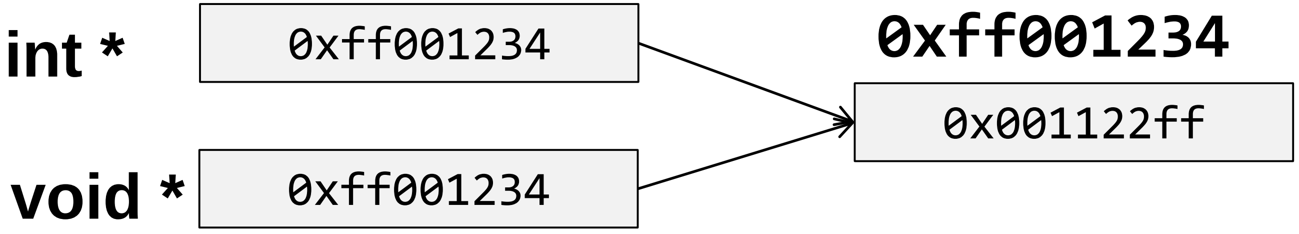
- Rules for automatic type conversions
  - Values of different types are first converted to the largest data type used in an expression, and then the expression is evaluated.
  - A value on the right side of an assignment statement is converted to the data type on the left side.
- Explicit cast:

```
int x=5;
double y;
y = (double)x;   // traditional C syntax
y = double(x);   // preferred C++ style
```

Type Conversion (II)

- Pointer assignment: change of how to interpret the data that the pointer variable points to.

```
int *intPointer;
void *genericPointer;
genericPointer = intPointer; // OK
/* in C, you can do this */
intPointer = genericPointer;
// in C++, you must do this
intPointer = (int *) genericPointer;
```



const Storage Type (I)

- #define should be replaced by constant variable in C++.
  - const variables must be initialized in declaration.

```
#define K_MAX_SIZE 10     /* in C */
const int kMaxSize=10;    // in C++
```

- Why constant variables are preferable?
  - Constant variables are visible to debuggers.
  - A constant variables has a type.

const Storage Type (II)

- const only modifies the type directly in front of it.
  - Scope: implicitly static to the file unit

```
int main() {
   char str1[] = "Hello";
   char str2[] = "Bye";
   const char * strPtr1 = str1;
   char * const strPtr2 = str1;
   strPtr1[0] = ‘T’; //illegal
   strPtr1 = str2;  //legal
   strPtr2[0] = ‘T’; //legal
   strPtr2 = str2;  //illegal
}
// const char * const str; ??? O.K.
```

Better I/O in C++

- The C++ way to do `printf`.

```
#include <iostream>
int x = 5;
double y = 6.0;
char *s = "Hello";
cout << x;
cout << y << '\n';
cout << s << '\n';
cout << "The value of x is " << x << " and
the value of y is " << y << ".\n";
```

output

```
56
Hello
The value of x is 5 and the value of y is 6.
```

Better I/O in C++ (Cont.)

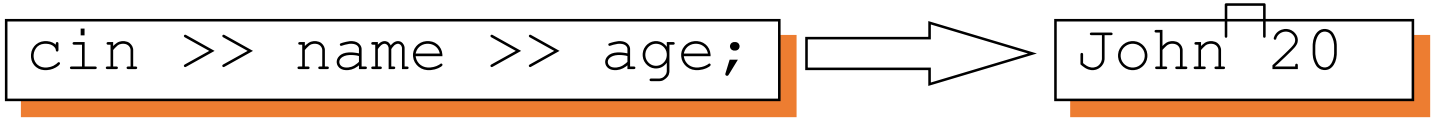
- The C++ way to do scanf.

```
#include <iostream>
int age;
char name[100];
cout << "What is your name?\n";
cin >> name;
cout << "How old are you?\n";
cin >> age;
cout<<"So, "<<name<<", you are "<<age<<"years old.";
```

output

```
What is your name?
John
How old are you?
20
So, John, you are 20 years old.
```

multiple cin:



```
cin >> name >> age;
```

```
John 20
```

Namespaces

- Scope resolution operator::

```
static int x=10;
void main() {
   int x=5;
   cout << x << "\n";
   cout << ::x << "\n";
}
```

```
Output:
5
10
```

- Namespace is used to group together logically related programming entities such as variables, objects, functions, and structures.

Example of Defining Namespaces

- Definition:

```
namespace Sample { // sample declaration
   int i;
   float f;
   void display() {cout<<i<<f;}
   float getf() {return f;}
}
namespace { // unnamed namespace declaration
   int i;
}
```

- Accessing namespace members

```
Sample::i=33;
float x=Sample::getf();
Sample::display();
i=22; //unamed namespace
```

Using Namespace

- Using namespace: using

```
#include <iostream>
using namespace std;
namespace Rectangle {
   float length;
   float width;
   void area() {cout<<“Area=“<<(length*width);}
}
using namespace Rectangle;
int main() {
   cout << “Enter length =>”; // std::cout;
   cin >> length; // std:cin;
   cout << “Enter width =>”; // std::cin;
   cin >> width; // std::cin;
   area(); // Rectangle::area();
   return 0;
}
```

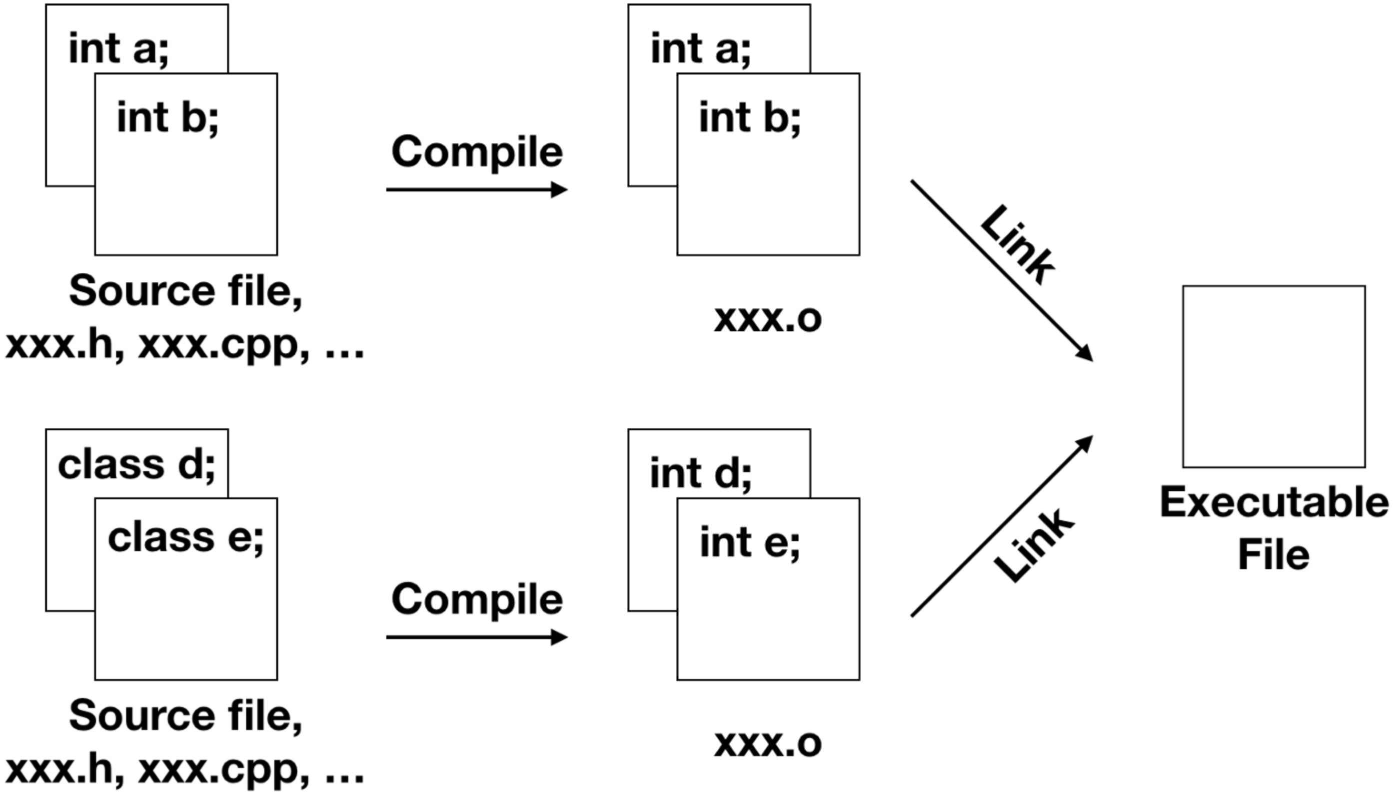
Nested namespace

```
#include<iostream>

int x = 20;
namespace outer
{
  int x = 10;
  namespace inner
  {
    int z = x; // this x refers to outer::x
  }
}

int main()
{
  std::cout<<outer::inner::z; //prints 10
  getchar();
  return 0;
}
```

Separate Compilation



int a;

int b;

Source file,
xxx.h, xxx.cpp, …

Compile

int a;

int b;

xxx.o

Link

class d;

class e;

Source file,
xxx.h, xxx.cpp, …

Compile

int d;

int e;

xxx.o

Link

Executable
File