Lecture 2

C++ Programming

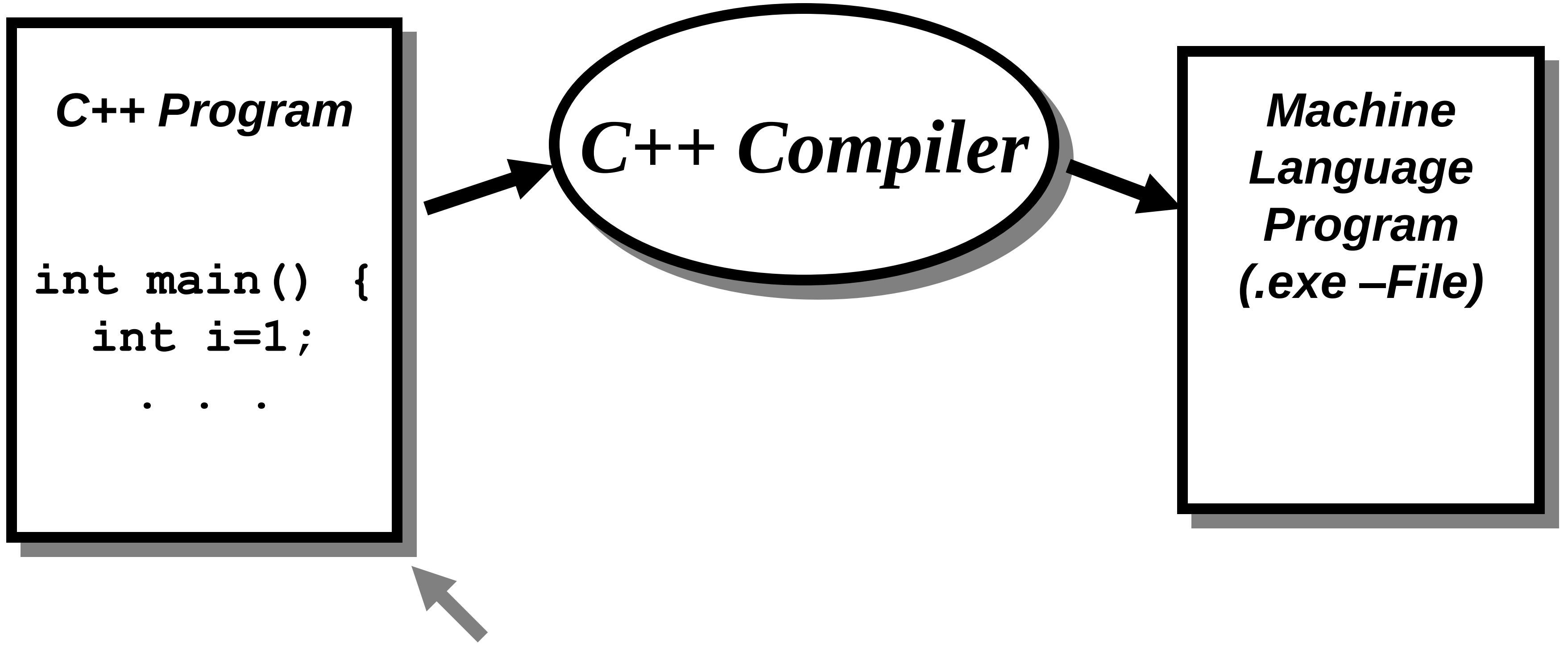
Arne Kutzner

Hanyang University / Seoul Korea

C++

- C++ is an extension of C.
- C++ was first proposed in the early 1980s
- Focus is on Object Oriented Programming
  - Data-centered form of programming
  - Objects have “attributes” and understand “methods”.
- C++ is an evolving programming language
  - Last ISO-accepted standard is C++20

Compiler



Created with text editor or development environment

Many Different Compilers

- There are many different C++ Compilers and Integrated Development Environments:
  - Microsoft Visual C++
    (Visual Studio Community:
    https://visualstudio.microsoft.com/downloads)
  - GCC (GNU g++)
    (Part of Linux distributions)
  - Clang (LLVM Project)
  - Intel C++ Compiler

C++ Compiler for Projects

- For this course I don't care what compiler/development environment you use as long as we can compile and run your programs.
- During class I will use
  Microsoft Visual C++ 2017
- For Apple user:
  Xcode 14: https://developer.apple.com/xcode/

Intro to C++ Language

Scalar Variables, Operators

and

Control Structures

Structure of a C++ Program

- A C++ program is a collection of declarations and definitions :
  - **data** declarations and definitions (local and global)
  - **function** declarations and definitions
  - ***class** declarations and definitions (OO-programming)*
  - a special start function called **`main()`**

Procedural vs. Object Oriented

- **Procedural languages** (C, Pascal etc.) express programs as a collection of functions/procedures .
- **Object Oriented languages** express programs as a collection of object types (called classes). (Different design principle!)
- **☹ C++ is both! ☹**
- *We will start with the procedural aspect of C++ and will later move to the Object Orientation within C++*

Hello World in C++

```
// Hello World program

#include <iostream>

int main() {

  std::cout << "Hello World\n";

  return 0;
}
```

comment

Allows access to an I/O library

Starts definition of special function main()

output (print) a string

Program returns a status code (0 means OK)

Comments

- Comments contain text that is not converted to machine language (it's just there for humans).
- Everything after `"//"` is ignored by the compiler.
- Everything between `"/*"` and `"*/"` is ignored.

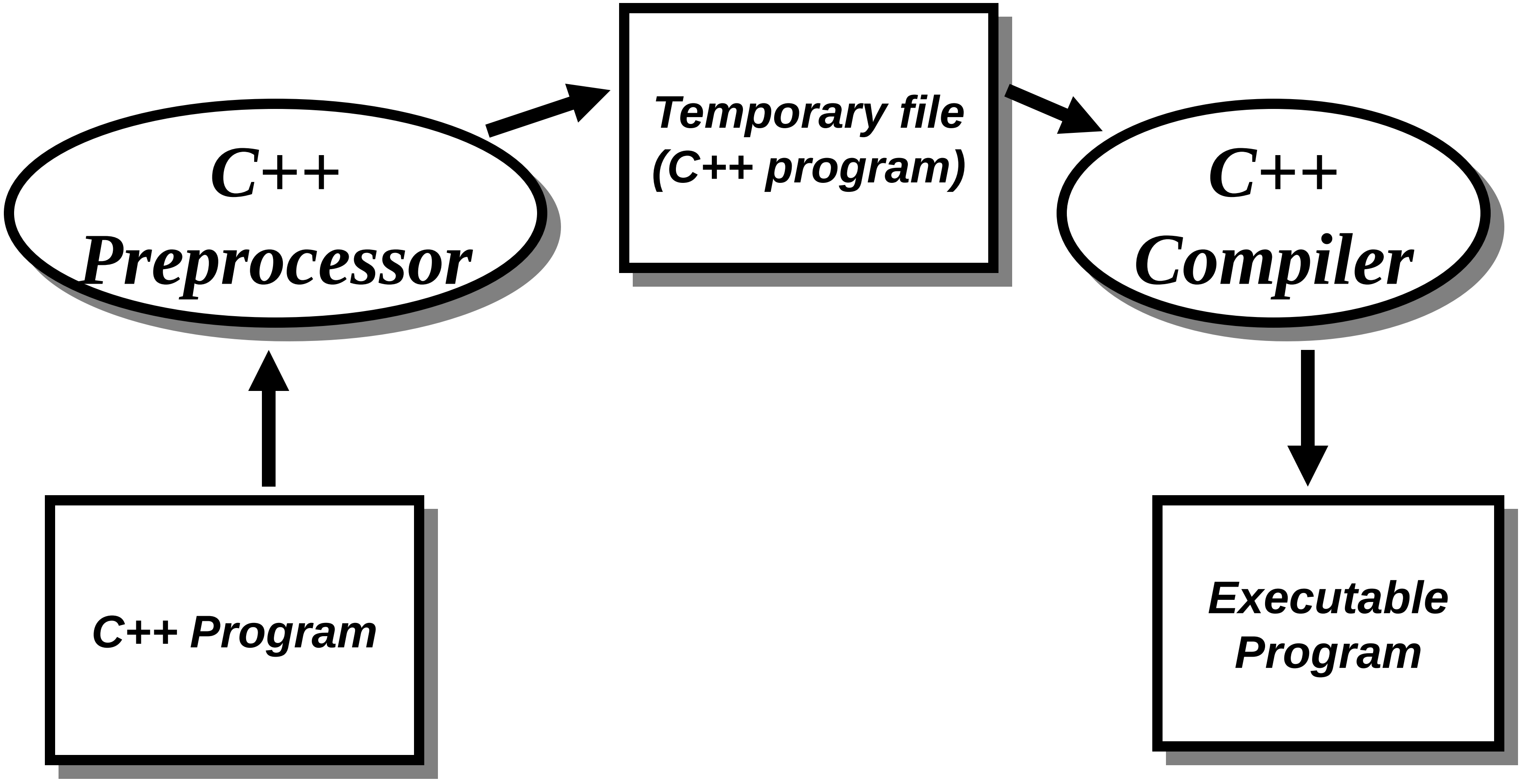
Comment Example

```
// Dave's Homework #1
// This program is awesome!
#include <iostream>
/* This program computes the
   coefficient of expansion of the
   universe to 27 decimal places.
*/
int main() {
  cout << 1.0000000000000000000000001;
}
```

C++ Preprocessor

- C++ Compilers automatically invoke a *preprocessor* that takes care of **`#include`** statements and some other special directives.
- You don't need to do anything special to run the preprocessor - it happens automatically.

Preprocessing



Preprocessor Directives

- **Preprocessor directives** are commands that give instructions to the C preprocessor, whose job it is to modify the program code before compilation

- Preprocessor directives always begin with a “#” character. Examples:

  `#include <`*`standard header file`*`>`

  `#define NAME `*`value`*

Includes

- The statement: **`#include <foo.h>`** inserts the contents of the file foo.h inside your file before the compiler starts.
- Definitions that allow your program to use the functions and classes that make up the standard C++ library are in these files.
- You can include your own file(s):
  `#include "myfile.h"`

#define (macro) Example
(C heritage)

```
#define NUM 45


y = NUM + NUM;
```

$\searrow$

becomes `y = 45 + 45;`

Parameterized **`#define`**
(C heritage)

- Macro definitions in C:

```
#define identifier(identifier,…
     ,identifier) token_string
```

Example macro definition:

```
#define SQUARE(x)  ((x) * (x))
```

Application:

`SQUARE(i+2)` expands to **`((i+2)*(i+2))`**

Examples of common includes

- Basic I/O (C++): `iostream`

- Standard Library (C): `stdlib.h`

- Time and Date support (C): `time.h`

C heritage

Variables

- *Variables* have to be declared/defined. Example:

```
int i, sum;
```

  - *Variables are names for locations in memory.*
- Variables must have a *type*
- Variables must be *declared* before they can be used.

Variables (cont.)

- Variables are declared like this:

  `type var_name;`

- *`type`* indicates what kind of variable.
- Built in types include:

  `int char float double bool`

- *You can also create new types. Later more ...*

Basic Data Types in C++

`char`	→	a single byte, capable of holding one character
`int`	→	an integer, typically reflecting the natural size of integer on the host machine
`float`	→	single precision floating-point
`double`	→	double precision floating-point
`bool`	→	a boolean value (true or false)

Variable Names

- C++ variable names:
  - made up of letters, digits and underscore.
  - Must start with a non-digit.
  - Case sensitive
    - `foo` is not the same name as `Foo`
- Can be any length
- *Good variable names tell the reader what the variable is used for!*

Little C++ Program …

```
// C++ Addition of integers
#include <iostream>
int main() {
  int integer1, integer2, sum;

  std::cout << "Enter first integer\n";
  std::cin >> integer1;
  std::cout << "Enter second integer\n";
  std::cin >> integer2;
  sum = integer1 + integer2;
  std::cout << "Sum is " << sum << std::endl;
  return 0;
}
```

Literals (Constants)

- Literals are fixed values used by a program.
- Some examples of literals:

```
22        3.14159        0x2A
false       "Hi Dave"       'c'
```

- You can initialize a variable in the declaration by *assigning* it a value:

```
int foo = 17;
double PI = 3.14159;
char c = 'a';
```

Expressions

- C++ *expressions* are used to express computation.
- Expressions include operations and the *operands* on which the operations are applied.
- Operands can be variables, literals or function calls.

Math Expressions

- Mathematical expressions have numeric values when evaluated.
- Some examples:

```
1+2
(fahr - 32)*(5/9)
1*(2*(3*(4*5)))
```

Mathematical Operators

`+    -    *    /    %`

- Operators have rules of *precedence* and *associativity* that control how expressions are evaluated.
- What is the value of this C++ expression ?:

```
2 / 3 / 4 + 5
```

- Answer: You can't tell unless you know the rules.

Associativity

- The associativity of an operator control the order of evaluation of expressions involving the same operator, for example:

```
3 / 4 / 5
```

- Associativity can be:
  - left-to-right: the leftmost operator is applied first.
  - Right-to-left: the rightmost operator is applied first.

Precedence

- Precedence controls the order of evaluation of operators.
  - A high precedence means an operator is evaluated (applied) before any lower precedence operators.
- In the case of different operators that have the same precedence C++ evaluates the left one first.

C++ Math Operator Rules

Operator	Associativity	Precedence
`()`	left to right	high
`* / %`	left to right	middle
`+ -`	left to right	low

- Now - what is the value of this?:

```
2 / 3 / 4 + 5
```

- How about this: `(7*3/4-2)*5`

Relational and Equality Operators

- Relational and Equality operators are used to compare values:
- Relational Operators:

>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal

- Equality Operators:

==	Equal to
!=	Not Equal to

Relational and Equality Operators (cont.)

- The relational operators have very low precedence and associate left-to-right.
- The equality operators have very-very low precedence and associate left-to-right.
- Some examples:

```
 17 < x              foo == 3.14
age != 21           x+1 >= 4*y-z
```

Another Operator

- The assignment operator "=" is used to assign a value to a variable.

```
x = 13 - y;
```

- Assignment has very low precedence and associates from right to left.
- You can do this:

```
x = y = z + 15;
```

Precedence

Operators	Precedence
`()`	highest (applied first)
`* / %`	
`+ -`	
`< <= > >=`	
`== !=`	
`=`	lowest (applied last)

Another Program

```
#include <iostream>
int main()
{
  double fahr,celcius;

  std::cout << "Enter Temperature in Fahrenheit\n";
  std::cin >> fahr;

  celcius = (fahr - 32.0)*5.0/9.0;
  std::cout << fahr << " fahrenheit is " << celcius
  << " Celcius" << std::endl;

  return 0;
}
```