

Lecture 3

TOC

1. C++ 簡介
2. Hello, World!
3. 變數與常數
4. 算術運算符
5. 比較運算符
6. 邏輯運算符
7. 位元運算符
8. Control Flow
9. Statement

C++ 簡介

簡介

C++ 由 Bjarne Stroustrup 於 1979 年開發，是 C 語言的超集，結合了 C 語言的效率和 OOP 的抽象能力。C++ 在 C 語言的基礎上，增加了類（Class）和模板（Template）等特性，使其成為一種面向對象的編程語言。

特點

- 支持面向對象的編程（OOP）
- 支持 C++ 的標準庫（STL）
- 在 C 語言的基礎上，增加了類（Class）和模板（Template）等特性
- 支持 C++ 的編譯器（g++）

Hello, World!

编写 C++ 程序

```
1  #include <iostream>
2
3  int main() {
4      std::cout << "Hello, World!\n";
5      return 0;
6  }
```

编译程序

```
g++ hello_world.cpp -o hello_world
./hello_world
```

编译选项

编译 C++ 程序时，可以使用一些编译选项来控制编译过程。例如，`-std=c++11` 用于指定 C++11 标准，`-g` 用于生成调试信息，`-O2` 用于启用优化。此外，还可以使用 `-I` 指定头文件的路径，使用 `-L` 指定库文件的路径。在代码中，`int, if, for, return` 是 C++ 的基本语法元素。

编译选项通常用于控制编译器的行为，例如生成可执行文件、库文件或中间文件。常见的编译选项包括 `-std`（指定 C++ 标准）、`-g`（生成调试信息）、`-O`（优化级别）、`-I`（头文件路径）、`-L`（库文件路径）等。在代码中，`int, if, for, return` 是 C++ 的基本语法元素。

□□□□□

- □□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□

```
extern int a; // declaration
int a = 10; // definition
```

```
void foo(); // declaration
void foo() { // definition
    // function body
}
```

```
class A; // declaration
class A { // definition
    // class body
};
```


□□

□□	□□□	□□□	□□□□
bool	bool	1	true/false
char	char	1	$-2^7 \sim 2^7 - 1$
short	short	2	$-2^{15} \sim 2^{15} - 1$
int	int	4	$-2^{31} \sim 2^{31} - 1$
long	long	4	$-2^{31} \sim 2^{31} - 1$
long long	long long	8	$-2^{63} \sim 2^{63} - 1$

unsigned 0 ~ $2^n - 1$ n signed

CPU/

C++11 int8_t, uint8_t, int16_t, uint16_t Reference

int $2^{31} - 1$

種類

種類	名前	サイズ	範囲
float	float	4	$3.4e-38 \sim 3.4e38$
double	double	8	$1.7e-308 \sim 1.7e308$
long double	long double	12	$3.4e-4932 \sim 1.1e4932$

IEEE 754

(Since C++23)

C++23 浮点数型 `float16_t`, `float32_t`, `float64_t` [Reference](#)

111

- $\pm\infty$: $\infty, -\infty$
- NaN(Not a Number): $0/0, \infty/\infty$ $\square\square$

```
std::numeric_limits<double>::infinity();
std::numeric_limits<double>::quiet_NaN();
std::numeric_limits<double>::signaling_NaN();
```

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

```
std::numeric_limits<T>::epsilon()
```

□ □ □ □ □

C++

void

void C
void*

- C-style cast: (type)expression
- C++ style cast: static_cast, dynamic_cast, const_cast, reinterpret_cast
-

```
int a = 10;  
float b = (float)a; // C-style cast  
double b = static_cast<double>(a); // C++ style cast
```

char → int → long → long long

-
- int int
- float → double → long double
-

```
#include <iostream>

int main() {
    char a= 127;
    char b = 3;
    int c = a + b;
    std::cout << c << std::endl;
    return 0;
}
```

C++ style cast v.s. C-style cast

- static_cast [][][][]
- static_cast [][][][] int [] char, [] reinterpret_cast [][][][]
[]
- C-style cast []

常量表达式

- 常量表达式变量使用 `constexpr` 声明

```
constexpr int x = 10;
constexpr int y = 10;
constexpr int sum = x + y; // 常量表达式
```

```
const int a = 10;
a = 20; // Error
```

- 常量表达式变量只能使用 `10, 3.14, 'a'` 等 C++ 常量表达式字面量

```
int a = 10; // 常量表达式
float b = 3.14; // 常量表达式
char c = 'a'; // 常量表达式
bool d = true; // 常量表达式
char e[] = "Hello, World!"; // 常量表达式
```

常量表达式变量只能使用 `'u, l, f'` 等常量表达式字面量

指针

指针(pointer) 存储内存地址的变量
引用(reference) 对变量的别名
std::nullptr_t 空指针常量

```
int a = 10;  
int* p = &a; // 指针  
int& r = a; // 引用  
  
int* p = nullptr; // 空指针  
*a = 11; // 非法  
r = 12; // 非法
```

sizeof

- sizeof(type)

sizeof

- sizeof(int)
- sizeof(char)

字符串

字符串的存储和表示 "Hello, World!" 字符串的存储和表示 \0 字符串的存储和表示

- 字符串的存储和表示 ASCII 字符串 UTF-8 字符串 UTF-16 字符串
- 字符串的存储和表示 \ 字符串的存储和表示 \n 字符串的存储和表示 \t 字符串的存储和表示 \\
- 字符串的存储和表示 R"()" 字符串的存储和表示 R"(Hello, \n World!)"

字符串 Unicode 字符串 C++11 字符串 u8 字符串 UTF-8 字符串 u8"?????" 字符串 wchar_t 字符串

구조체(C struct)

구조체는 C 언어에서 데이터의 집합을 정의하고, 이 집합을 변수에 할당하여 사용하는 방법이다.

```
struct Student {  
    int id;  
    char name[20];  
    float score;  
};  
  
struct Class {  
    int id;  
    Student students[30];  
};  
  
struct Student s = {1, "Alice", 90.5};  
struct Student t = {.id = 1, .name = "Alice", .score = 90.5};  
struct Class c = {1, {s, t}};  
  
// access struct member  
std::cout << s.id << s.name << std::endl;
```

Union(联合体)

联合体是C语言中一种特殊的数据类型，它允许在相同的内存位置存储不同的数据类型。在C++中，联合体通常用于实现变长模板（variant）。

```
Union U {  
    int a;  
    double b;  
};  
  
size_t size = sizeof(U); // 8  
  
U u;  
u.a = 10;  
std::cout << u.a << std::endl; // 10  
u.b = 3.14;  
std::cout << u.b << std::endl; // 3.14
```

C++17 引入了 `std::variant` 来模拟 C 语言中的 Union 联合体。

```
enum(□□)
```

```
static_cast<T>()
```

```
#include <iostream>

int main() {
    enum Color {
        RED,
        GREEN,
        BLUE
    };

    enum Weather {
        SUNNY = 10,
        RAINY = 20,
        CLOUDY = 30
    };

    Color c = RED;
    Weather w = SUNNY;

    std::cout << c << '\n' << w << std::endl;
}
```

`C` `C++11` `(enum class)`

Control Flow

□□□□

if □□

```
if (condition) {  
    // code block  
} else if (condition) {  
    // code block  
} else {  
    // code block  
}
```

switch □□

```
switch (expression) {  
    case constant1:  
        // code block  
        break;  
    case constant2:  
        // code block  
        break;  
    default:  
        // code block  
}
```

□□□□

while □□

```
while (condition)
    // statement

do {
    // code block
} while (condition);
```

for □□

```
for (int i = 0; i < 10; i++)
    // statement

int b = 10;
for (; b > 0; b--)
    // statement
```

break □□□□□□□□ continue □□□□□□□□□□

goto: □□□□□□□□□□□□□□ C++ □□□□□□□□

Statement

□□□□□□□□ C++ □□□□,□□□□□□□□

1. □□□□
2. □□□□□
3. □□□□□
4. □□□□
5. □□□□
6. □□□□
7. □□□□
8. try □

```
// {} □□□□□□□□  
{  
    int a = 1;  
    std::cout << a << std::endl;  
}
```