

lecture3

Lecture 3: Initialization and References

what: Initialization and References

what more:

initialization

references

L/R-values

const

compiling c++ programs

1. Initialization (初始化)

| **Definition:** Provides initial values at the time of construction.

↳ [2025Fall-03-InitializationAndReferences,_p.8](#)

| | What?: “Provides initial values at the time of construction” - cppreference.com

A. Direct Initialization

- 语法: 使用圆括号 ()。
- 示例: `int num(12.0);` -> 输出为 12。
- 风险: C++ 不会进行类型检查, 会发生 **Narrowing Conversion** (收窄转换/数据截断), 且不会进行四舍五入。

B. Uniform Initialization (C++11)

- 语法: 使用花括号 {}。
- 优点:
 - **Safe:** 不允许收窄转换, 类型不匹配时会报错 (例如 `int num{12.0}` 会触发 Error)。
 - **Ubiquitous:** 适用于所有类型, 如 `std::vector<int> num{1,2,3}` 或 `map`。

C. Structured Binding (C++17)

- 语法: 类似于 `auto [x, y, z] = ...;`
- 作用: 能够同时访问并初始化多个返回值。
- 代码示例: C++

```

auto [className, buildingName, language] = getClassInfo();

#include <iostream>
#include <tuple>
#include <string>

std::tuple<std::string, std::string, std::string> getClassInfo() {
    std::string className = "CS106L";
    std::string buildingName = "Thornton 110";
    std::string language = "C++";
    return {className, buildingName, language};
}

int main() {
    auto [className, buildingName, language] = getClassInfo();
    std::cout << "Come to " << buildingName << " and join us for " << className
        << " to learn " << language << "!" << std::endl;

    return 0;
}

```

2. References (引用)

- 本质: 已存在变量的一个 **Alias** (别名)。
- 语法: 使用 & 符号。
- 示例: C++

```

int num = 5;
int &ref = num; // ref 现在是 num 的别名
ref = 10;       // num 的值也变为了 10

```

3. L-values and R-values (左值与右值)

特性	L-value (左值)	R-value (右值)
全称	Locator Value	Read Value
位置	等号左侧或右侧	只能在等号右侧
内存	有明确的内存地址	临时变量, 无内存地址
示例	int x = 10; (x 是左值)	int x = 10; (10 是右值)



关键限制

引用只能绑定到左值 (**L-values**)。例如: `void squareN(int &n)` 调用 `squareN(5)` 会报错, 因为 5 是一个右值。

4. Const 限定符

| **Definition:** 声明对象不可被修改。

- `vector<int> vec`: 可读、可修改。
- `const vector<int> const_vec`: 仅可读。
- `vector<int>& ref_vec`: 原变量的新别名 (可读写)。
- `const vector<int>& const_ref`: 仅可读窗口, 禁止通过此引用修改。

const

```
#include <iostream>
#include <vector>

int main()
{
    std::vector<int> vec{ 1, 2, 3 };    /// a normal vector
    const std::vector<int> const_vec{ 1, 2, 3 };    /// a const vector
    std::vector<int>& ref_vec{ vec };    /// a reference to 'vec'
    const std::vector<int>& const_ref{ vec };    /// a const reference

    vec.push_back(3);    /// this is ok!
    const_vec.push_back(3);    /// no, this is const!
    ref_vec.push_back(3);    /// this is ok, just a reference!
    const_ref.push_back(3);    /// this is const, compiler error!

    return 0;
}
```

5. Compiling C++ Programs (程序编译)

编译命令:

```
g++ -std=c++23 main.cpp -o main
```

- `g++`: 编译器命令。
- `-std=c++23`: 指定 C++ 版本标准。

- `main.cpp` : 源文件。
- `-o` : 指定输出文件名 (如果不加, 默认生成 `a.out`)。
- `main` : 生成的可执行文件名。

A recap of today!

- Use uniform initialization — it works for all types and objects!
- References are a way to alias variables!
- You can only reference an l-value!
- **const** is a way to ensure that you can't modify a variable