Initialization & References

Initialization

1. Direct initialization 直接初始化 =
2. Uniform initialization 统一初始化 {}
3. Structured Binding 结构化绑定

```
#include <iostream>
//统一初始化
int main() {
 // Notice the brackets
 int numOne{12.0};
 float numTwo{12.0};
 std::cout << "numOne is: " << numOne << std::endl;
 std::cout << "numTwo is: " << numTwo << std::endl;
 return 0;
}
```

```
//结构化绑定:
//解构出多个值，赋值给多个变量
#include <iostream>
#include <tuple>
#include <string>
std::tuple<std::string, std::string, std::string> getClassInfo() {
    std::string className ="CS106L";
    std::string buildingName ="Turing Auditorium";
    std::string language ="C++";
    return {className, buildingName, language};
}
int main() {
    auto classInfo = getClassInfo();
    std::string className = std::get<0>(classInfo);
```

```
    std::string buildingName = std::get<1>(classInfo);
    std::string language = std::get<2>(classInfo);
    std::cout << "Come to " << buildingName << " and join us for "
<< className
    << " to learn " << language << "!" << std::endl;
    return 0;
}
```

2.References

- &函数引用参数能够修改值
- ```
  void squareN(int& n) {
   // calculates n to the power of 2
   n = std::pow(n, 2);
  }
  ```

- 使用auto捕获需要加上&，否则仅仅是创建了一个副本，而不是对元素进行引用
- ```
  void shift( vector< pair<int, int> > &nums) {
      for (auto &[num1,num2] : nums) {
          num1++,num2++;
      }
  }
  ```

3.L-values vs R-values

- 左值是是指**有明确存储位置**的值
- 右值是**不具有持久存储位置**的值，通常是个常数或者临时对象
- 一个小小的问题：这里的int& num是什么?
- ```
  int squareN(int& num) {
   return std::pow(num, 2);
  }
  ```

- 很明显，这里引用了地址所以是L-Value左值
- 注意，无法对右值直接引用
  ```

- ```
  int squareN(int& num) {
      return std::pow(num, 2);
  }
  int main()
  {
      int lValue = 2;
      auto four = squareN(lValue);
      auto fourAgain = squareN(2);//这里的引用会报错
      return 0;
  }
  ```

- 但是c++11可以通过使用&&来对右值引用
- ```
  int squareN(int& num) {
      return std::pow(num, 2);
  }
  int squareN_2(int&& num) {
      return std::pow(num, 2);
  }
  int main()
  {
      int lValue = 2;
      auto four = squareN(lValue);
      auto fourAgain = squareN_2(2);
      //这样不会报错
      return 0;
  }
  ```

- 还可以使用***const int& num***
- ```
  int squareN(const int& num) {
      // 使用 const 引用，支持左值和右值
      return std::pow(num, 2);
  }
  ```

4.Const

- 一个简单的示范 对于const的vector而言
  ```

- ```
  int main()
  {
      std::vector<int> vec{ 1, 2, 3 }; // 一个普通的 vector
      const std::vector<int> const_vec{ 1, 2, 3 }; // 一个常量
  vector

      std::vector<int>& ref_vec{ vec }; // 对 'vec' 的引用
      const std::vector<int>& const_ref{ vec }; // 对 'vec' 的
  常量引用

      vec.push_back(3); // 这可以，vec 是可修改的!
      const_vec.push_back(3); // 错误，这个是常量!
      ref_vec.push_back(3); // 这只是引用!
      const_ref.push_back(3); // 是常量引用，编译错误!

      return 0;
  }
  ```

- 演示对于一个const变量的复制
- ```
      const std::vector<int> const_vec{ 1, 2, 3 };
      std::vector<int>& bad_ref{ const_vec }; /// BAD
  //  错误的示范
  ```

- ```
  const std::vector<int> const_vec{ 1, 2, 3 };
  const std::vector<int>& bad_ref{ const_vec };
  //正确的示范
  ```

5.Compiling C++ programs

- cpp是编译语言
- 编译器有 clang and g++
- ```
  //如何使用g++编译的演示
  g++ -std=c++11 <source file> -o <out_name>
  ```