Task8 Lambda functions & Smart pointer

Lambda functions

- The most common lambda expression is something like

  ```
  auto plus = [] (int v1, int v2) -> int { return v1 + v2; }
  int sum = plus(1, 2);
  ```

  However, just as Kong Yiji said "*There are four ways to write fennel beans*", there are also four ways to write lambda expressions.

  Study the official C++ documentation on your own and briefly introduce these four ways.

- Please read the following code.

  ```
  #include <iostream>
  #include <functional>
  int main() {
      int a = 5;
      int b = 10;
      auto f = [a, &b]() mutable -> int {
          std::cout << "Inside lambda - Before: a=" << a << ", b=" << b <<
  std::endl;
          a += 5;
          b += 5;
          std::cout << "Inside lambda - After: a=" << a << ", b=" << b <<
  std::endl;
          return a + b;
      };
      int result = f();
      std::cout << "Outside lambda - a=" << a << ", b=" << b << std::endl;
      std::cout << "Result of lambda: " << result << std::endl;
      return 0;
  }
  ```

  What will be the output of this code?

  If the `mutable` keyword is removed, will it produce the same results, and why?

- Please read the following code.

  ```
  struct Item
  {
      Item(int aa, int bb) : a(aa), b(bb) {}
      int a,b;
  };
  int main()
  {
      std::vector<Item> vec;
  ```

```
    vec.push_back(Item(1, 19));
    vec.push_back(Item(10, 3));
    vec.push_back(Item(3, 7));
    vec.push_back(Item(8, 12));
    vec.push_back(Item(2, 1));

    // 根据Item中成员a升序排序
    std::sort(vec.begin(), vec.end(),lambda1);

    // 打印vec中的item成员
    std::for_each(vec.begin(), vec.end(),lambda2);
    return 0;
}
```

Among them, `lambda1` and `lambda2` are two lambda functions you need to implement.

Please give their implementation according to the requirements of the comments.

Smart pointers

- C++11 introduced 3 types of smart pointers:

  std::unique_ptr: A pointer with exclusive ownership of resources.
  std::shared_ptr: A pointer that shares ownership of resources.
  std::weak_ptr: An observer of shared resources, to be used with std::shared_ptr, does not affect the lifecycle of resources.

  Explain the meaning of "resource ownership" here.

- Self-study https://zhuanlan.zhihu.com/p/150555165, master and introduce the basic usage of the three smart pointers.

- Smart pointers in C++ are a typical application of the RAII concept. For example, std::shared_ptr, std::unique_ptr, etc., they acquire resources during construction and release resources during destruction to ensure that there is no memory leak.

  What is the RAII concept? What is the significance of the RAII concept for project development?

- Please read the following code.

```
#include <iostream>
#include <memory>

class MyObject {
public:
    MyObject(int id) : id_(id) {
        std::cout << "MyObject " << id_ << " constructed.\n";
    }
    ~MyObject() {
        std::cout << "MyObject " << id_ << " destructed.\n";
    }
    void show() {
```

```cpp
        std::cout << "This is MyObject " << id_ << ".\n";
    }
private:
    int id_;
};

void process(std::unique_ptr<MyObject> ptr) {
    ptr->show();
}

int main() {
    // 使用make_unique生成一个std::unique_ptr实例
    std::unique_ptr<MyObject> objPtr = std::make_unique<MyObject>(1);
    // 通过->访问对象成员函数
    objPtr->show();

    // 使用std::move转移所有权到函数process
    process(std::move(objPtr));

    // 检查ptr是否为空
    if (!objPtr)
        std::cout << "objPtr is empty.\n";

    return 0;
}
```

Explain what happens during the execution of this piece of code.

What would happen if the `unique_ptr` in this code were replaced with the other two types of smart pointers, respectively?