现代C++题目

卢瑟帝国

2023 年 11 月 9 日

暂时只有 13 道题目，并无特别难度，有疑问可看视频教程或答案解析。

1 实现管道运算符

日期：2023/7/21 出题人：mq白

给出以下代码，在不修改已给出代码的前提下使它满足**运行结果**。

```
int main(){
    std::vector v{1, 2, 3};
    std::function f {[](const int& i) {std::cout << i << ' '; } };
    auto f2 = [](int& i) {i *= i; };
    v | f2 | f;
}
```

要求运行结果

```
1 4 9
```

- **难度：** ★★☆☆☆
  提示： `T& operator|(T& v, const T& f)` 。

2 实现自定义字面量 _f

日期：2023/7/22 出题人：mq白

给出以下代码，在不修改已给出代码的前提下使它满足**运行结果**。 6 为输入，决定 π 的小数点后的位数，可自行输入更大或更小数字。

```cpp
int main(){
    std::cout << "乐 :{} *\n"_f(5);
    std::cout << "乐 :{0} {0} *\n"_f(5);
    std::cout << "乐 :{:b} *\n"_f(0b01010101);
    std::cout << "{:*<10}"_f("卢瑟");
    std::cout << '\n';
    int n{};
    std::cin >> n;
    std::cout << "π : {:.{}f}\n"_f(std::numbers::pi_v<double>, n);
}
```

要求运行结果

```
乐 :5 *
乐 :5 5 *
乐 :1010101 *
卢瑟******
6
π : 3.141593
```

- **难度：** ★ ★ ☆ ☆ ☆

  提示：C++11 用户定义字面量、C++20 format 库。

3 实现 print 以及特化 std::formatter

日期：2023/7/24 出题人：mq白

实现一个 print，如果你做了上一个作业，我相信这很简单。要求调用形式为:

```
print(格式字符串, 任意类型和个数的符合格式字符串要求的参数)
```

```cpp
struct Frac {
    int a, b;
 };
```

给出自定义类型Frace，要求支持以下：

```
Frac f{ 1,10 };
print("{}", f);// 结果为1/10
```

要求运行结果

1/10

- 难度： ★ ★ ★ ☆ ☆

  提示：std::formatter。

禁止面向结果编程，使用宏等等方式，本题主要考察和学习 format 库，记得测试至少三个不同编译器。

4 给定模板类修改，让其对每一个不同类型实例化有不同 ID

日期：2023/7/25 出题人：Maxy

```
#include <iostream>
class ComponentBase{
protected:
    static inline size_t component_type_count = 0;
};
template<typename T>
class Component : public ComponentBase{
public:
    //todo...
    //使用任意方式更改当前模板类，使得对于任意类型X，若其继承自Component

    //则X::component_type_id()会得到一个独一无二的size_t类型的id（对于不同的X类型返回的值应不同）
    //要求：不能使用std::type_info（禁用typeid关键字），所有id从0开始连续。
};
class A : public Component<A>
{};
class B : public Component<B>
{};
class C : public Component<C>
```

```
{};
int main()
{
    std::cout << A::component_type_id() << std::endl;
    std::cout << B::component_type_id() << std::endl;
    std::cout << B::component_type_id() << std::endl;
    std::cout << A::component_type_id() << std::endl;
    std::cout << A::component_type_id() << std::endl;
    std::cout << C::component_type_id() << std::endl;
}
```

要求运行结果

```
0
1
1
0
0
2
```

- 难度： ★ ☆ ☆ ☆ ☆
  提示：初始化。

5 实现 scope_guard 类型

日期：2023/7/29 出题人：Da'Inihlus

要求实现 **scope_guard** 类型（即支持传入任意可调用类型，析构的时候同时调用）。

```
#include <cstdio>
#include <cassert>

#include <stdexcept>
#include <iostream>
#include <functional>

struct X {
    X() { puts("X()"); }
```

```
10      X(const X&) { puts("X(const X&)"); }
11      X(X&&) noexcept { puts("X(X&&)"); }
12      ~X() { puts("~X()"); }
13  };
14
15  int main() {
16      {
17          // scope_guard的作用之一，是让各种C风格指针接口作为局部变量时也能得到RAII支持
18          // 这也是本题的基础要求
19          FILE * fp = nullptr;
20          try{
21              fp = fopen("test.txt","a");
22              auto guard = scope_guard([&] {
23                  fclose(fp);
24                  fp = nullptr;
25              });
26
27              throw std::runtime_error{"Test"};
28          } catch(std::exception & e){
29              puts(e.what());
30          }
31          assert(fp == nullptr);
32      }
33      puts("----------");
34      {
35          // 附加要求1，支持函数对象调用
36          struct Test {
37              void operator()(X* x) {
38                  delete x;
39              }
40          } t;
41          auto x = new X{};
42          auto guard = scope_guard(t, x);
43      }
44      puts("----------");
45      {
46          // 附加要求2，支持成员函数和std::ref
```

```
47          auto x = new X{};
48          {
49              struct Test {
50                  void f(X*& px) {
51                      delete px;
52                      px = nullptr;
53                  }
54              } t;
55              auto guard = scope_guard{&Test::f, &t, std::ref(x)};
56          }
57          assert(x == nullptr);
58      }
59  }
```

要求运行结果

```
Test
————-
X()
X()
————-
X()
X()
```

- 难度： ★ ★ ★ ★ ☆

  提示：C++11 形参包，成员指针，完美转发，std::tuple，std::apply，C++17 类推导指引，std::invoke，std::function

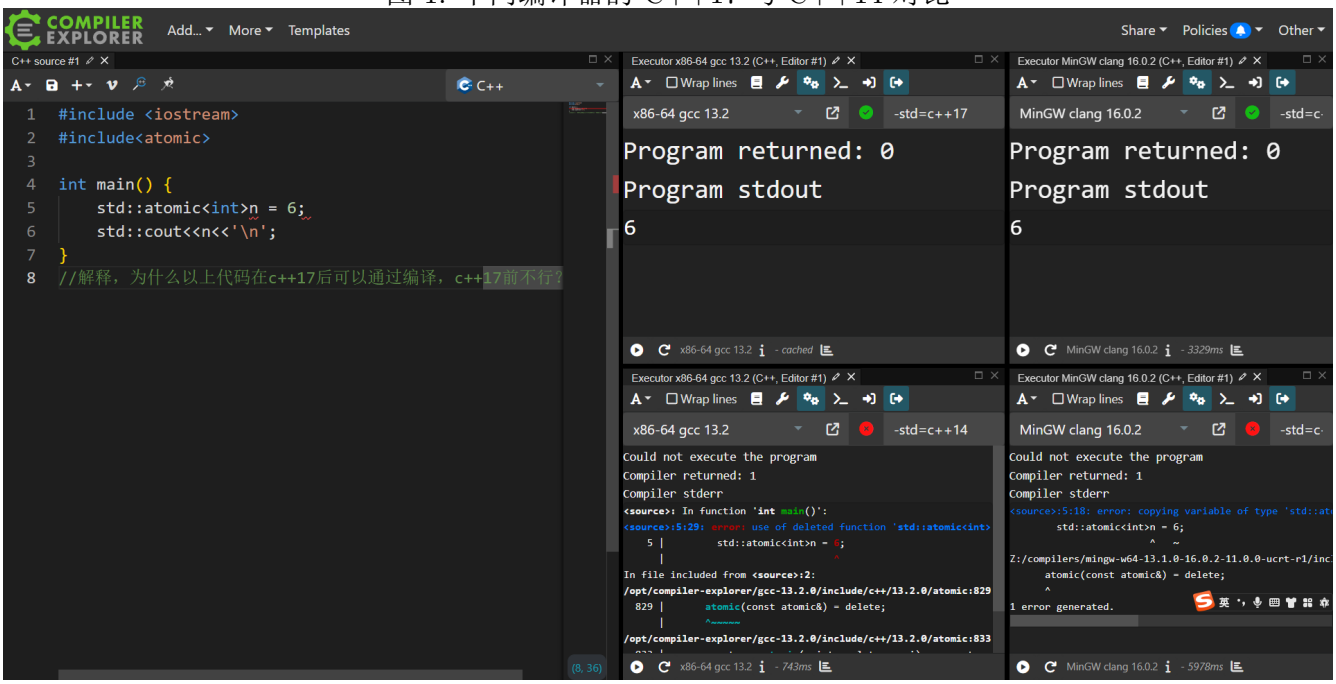
6 解释 std::atomic 初始化

日期：2023/8/2 出题人：mq白

```
1  #include <iostream>
2  #include <atomic>
3  int main() {
4      std::atomic<int> n = 6;
5      std::cout << n << '\n';
6  }
```

详细解释，为什么以上代码在 C++17 后可以通过编译， C++17 前不行？

图 1: 不同编译器的 C++17 与 C++14 对比



- 难度： ★ ★ ★ ☆ ☆

  提示：复制消除。

7 throw new MyException

日期：2023/8/6 出题人：mq白

给出代码：

```
struct MyException :std::exception {
    const char* data{};
    MyException(const char* s) :data(s) { puts("MyException()"); }
    ~MyException() { puts("~MyException()"); }
    const char* what()const noexcept { return data; }
   };
   void f2() {
    throw new MyException("new Exception异常....");
   }
   int main(){
```

```
11          f2();
12      }
```

灵感来源自 Java 人写 C++。

在 main 函数中自行修改代码，接取 f2() 函数抛出的异常（try catch）。

要求运行结果

```
MyException()
new Exception异常....
MyException()
```

- **难度：** ★ ☆ ☆ ☆ ☆

  提示：std::exception，try catch

8 定义 array 推导指引

日期：2023/8/12 出题人：mq白

给出代码：

```
template<class Ty,size_t size>
struct array {
    Ty* begin() { return arr; };
    Ty* end() { return arr + size; };
    Ty arr[size];
};
int main() {
    ::array arr{1, 2, 3, 4, 5};
    for (const auto& i : arr) {
        std::cout << i << ' ';
    }
}
```

要求自定义推导指引，不更改已给出代码，使得代码成功编译并满足运行结果。

要求运行结果

```
1 2 3 4 5
```

- **难度：** ★ ★ ★ ☆ ☆
  提示：参考 std::array 实现，C++17类模板推导指引

9 名字查找的问题

日期：2023/8/15 出题人：mq白

```cpp
#include<iostream>

template<class T>
struct X {
    void f()const { std::cout << "X\n"; }
};

void f() { std::cout << "全局\n"; }

template<class T>
struct Y : X<T> {
    void t()const {
        this->f();
    }
    void t2()const {
        f();
    }
};

int main() {
    Y<void>y;
    y.t();
    y.t2();
}
```

给出以上代码，要求解释其运行结果。

要求运行结果

```
X
全局
```

- **难度：** ★ ★ ★ ☆ ☆

  提示：名字查找。本问题堪称经典，在某著名 template 书籍也有提过（虽然它完全没有讲清楚）。并且从浅薄的角度来说，本题也可以让你向其他人证明加 this 访问类成员，和不加，是有很多区别的。

10 遍历任意聚合类数据成员

题目的要求非常简单，在很多其他语言里也经常提供这种东西（一般是反射）。但是显而易见 C++ 没有反射。

我们给出代码：

```
int main() {
    struct X { std::string s{ " " }; }x;
    struct Y { double a{}, b{}, c{}, d{}; }y;
    std::cout << size<X>() << '\n';
    std::cout << size<Y>() << '\n';

    auto print = [](const auto& member) {
        std::cout << member << ' ';
    };
    for_each_member(x, print);
    for_each_member(y, print);
}
```

要求自行实现 for_each_member 以及 size 模板函数。要求支持任意自定义类类型（聚合体）的数据成员遍历（聚合体中存储数组这种情况不需要处理）。这需要打表，那么我们的要求是支持聚合体拥有 0 到 4 个数据成员的遍历。

要求运行结果

```
1
4
0 0 0 0
```

- **难度：** ★ ★ ★ ★ ☆

  提示：学习，boost::pfr。

11 emplace_back() 的问题

日期：2023/8/20 出题人：jacky

思考：以下代码为什么在 C++20 以下的版本中无法成功编译，而在 C++20 及以后却可以？

```cpp
#include <vector>

struct Pos {
    int x;
    int y;
};

int main(){
    std::vector<Pos> vec;
    vec.emplace_back(1, 5);
}
```

- 难度： ★ ★ ☆ ☆ ☆

  提示：new，聚合初始化。

12 实现make_vector()

日期：2023/8/28 出题人：jacky

请实现函数 make_vector(...)，使以下代码编译通过（C++20）：

```cpp
#include <cstdio>
#include <vector>

inline void dbg(const char* msg)
{
    std::puts(msg);
    std::fflush(stdout);
}

struct X {
```

```
11      X() noexcept
12      {
13          dbg("X()");
14      };
15
16      ~X() noexcept
17      {
18          dbg("~X()");
19      };
20
21      X(const X&)
22      {
23          dbg("X(const X&)");
24      }
25
26      X(X&&) noexcept
27      {
28          dbg("X(X&&)");
29      }
30  };
31
32  void test()
33  {
34      static_assert(requires {
35          {
36              make_vector(std::vector{1, 2, 3})
37          } -> std::same_as<std::vector<std::vector<int>>>;
38          {
39              make_vector(1, 2, 3)
40          } -> std::same_as<std::vector<int>>;
41          make_vector(1, 2, 3).size() == 3;
42      });
43      X    x1;
44      X    x2;
45      auto vec = make_vector(x1, std::move(x2));
46  }
47
```

```
48  int main()
49  {
50      test();
51      dbg("test end");
52  }
```

要求运行结果

```
X()
X()
X(const X&)
X(X&&)
X(const X&)
X(const X&)
X()
X()
X()
X()
X()
X()
test end
```

- 难度： ★ ★ ★ ☆ ☆
  提示：重载决议

13 关于 return std::move(expr)

日期：2023/9/6 出题人：mq白

我们会给出三段使用到了 `return std::move(expr)` 代码。
解释说明这些代码是否有问题，问题在哪，或者没问题，那么为什么要这样使用。

1. 全局函数，返回局部对象，使用 std::move。

```
1       #include<iostream>
2
3       struct X{//后续代码不再重复X类
4           X() { puts("X()"); }
```

```
5           X(const X&) { puts("X(const X&)"); }
6           X(X&&)noexcept { puts("X(X&&)"); }
7           ~X() { puts("~X()"); }
8       };
9
10      X f(){
11          X x;
12          return std::move(x);
13      }
14
15      int main(){
16          X x = f();
17      }
```

2. 全局函数，返回局部的引用，使用 std::move。

```
1       X&& f(){
2           X x;
3           return std::move(x);
4       }
```

3. 类中成员函数，返回数据成员，使用 std::move。

```
1       struct Test {
2           X x;
3           X f() {
4               return std::move(x);
5           }
6       };
```

- **难度：** ★ ★ ★ ☆ ☆
  提示：return 重载决议。