Homework 5 客观题 (a checklist for C++ beginners)

12(错)

太傻逼了，怎么可能 `sizeof` 是在运行时决定的呢？它肯定是在编译时就决定的（除非它是 VLA）

18（错）

`const_cast<>` 是可以去除 `low-level const`

Homework 6 客观题

2

指针相减：指向同一块连续内存的时候可以相减

ps：可以指向尾后位置（最后一个位置的后一个位置）

```
Type a[10];
Type *p1 = a + i, *p2 = a + j;
```

`p1 - p2` 的结果是 `i - j` ，至于 `sizeof` 在需要的时候编译器会自动帮你乘上去

3

`delete` 函数会参与重载决议，这意味着你有可能调用一个 `deleted` 函数，如下：

```
void fun(int) = delete;
void fun(double) {
  std::cout << "fun(double) called.\n";
}
int ival = 42;
fun(ival);
```

运行结果是：报错并表明你在调用一个 deleted 的 function

4

加上 `typedef` 你在创建一个类型别名
去掉 `typedef` 你在定义一个变量

5 (错)

A constructor（and the destructor）has no return type！！！！！
你可以返回空，但不可以返回任何类型的变量\值

7（错）

```
class A {
  int arr[100];

 public:
  void print() const {
    for (auto i = 0; i != 100; ++i)
      std::cout << arr[i] << ' ';
    std::cout << std::endl;
  }
};

int main() {
  A a;          //default-initialization
  A b{};        //value-initialization
  a.print();
  b.print();
}
```

其实，value-initialization **并不一定意味着** default-initialization!

7（错）

如果没有 non-trivial default-constructor, value-initializor 会 zero initialize 所有的成员。

什么时候 default-constructor 是 non-trivial 的?

- 不可以是 user-provided
  - 如果默认构造函数后为空函数体，则是 non-trivial 的，value-initializor 会调用这个默认构造函数，这意味着永远不会零初始化（除非你人为定义了）
- 有一个不可以 default initialize 的成员（比如 std::string）

在这种情况（有 non-trivial default-constructor），两者可以理解为是一样的。

8

`new int [0]` 会不会分配内存?

会!

`new` 不会返回空指针，如果内存分配失败，它会抛出异常！（与 `malloc` 行为不相同）

10（错）

```
class Dynarray {
  int *m_storage;
  std::size_t m_length;
 public:
  int &at(std::size_t n) const {
    return m_storage[n];
  }
};
```

`int & at` 对吗？ 为什么不是 `const int &` ？

`const` 作用于隐式的 `this` 指针上，`m_storage` 通过 `this` 取值，所以 `m_storage` 确实是 `const` 指针，但是它真正的类型是 `int * const` ，也就是说它其实是一个顶层 `const` 指针，所以它指向的对象是 `int &` 而不是 `const int &` ，所以你也可以通过 `at()` 来修改 `m_storage` 中的值。

10（错）

但是数组并不像指针！

对于数组而言，如果它内部的元素是 `const`，那么它本身也应该是 `const` 类型。
但是对于指针而言，它的 底层 `const` 和 顶层 `const` 并没有直接的关联。

12（错）

别忘了，`delete` 只能销毁 `new` 定义的指针！！！

14

copy-and-swap

引用折叠

引用的引用折叠为引用

16（错）

call by value 可以做到：
拷贝左值，移动右值，如果不存在移动操作，则拷贝右值。

`std::move` 做了什么？

- 将一个左值的值类型转化为右值（xvalue），返回一个 xvalue.