选择题 第六次

选择题

刷新

如无特殊说明，所有题目的编译选项都包含 `-std=c++11`

单选题

1. 以下说法正确的是:

   A. 一个裸指针可以初始化多个智能指针，这几个智能指针将共享同一个辅助指针

   B. 不能将多个智能指针指向同一个对象

   C. 使用 类对象的指针 `ptr` 构造 `shared_ptr` 后，则 `delete ptr;` 是合法操作，不会造成潜在的问题

   D. 不能直接使用智能指针维护对象数组

2. 以下选项正确的是：

```
std::string text = "I have 2 apples, 10 oranges, and 5 bananas.";
std::regex pattern("(\\d+)\\s+(\\w+?)");
std::smatch matches;
std::vector<std::string> fruits;

std::sregex_iterator iter(text.begin(), text.end(), pattern);
std::sregex_iterator end;

while (iter != end) {
    matches = *iter;
    std::cout << matches[2] << ": " << matches[1] << std::endl;
    ++iter;
}
```

   A. 输出结果：

```
apples: 2
oranges: 10
bananas: 5
```

   B. 输出结果：

```
2: apples
10: oranges
5: bananas
```

C. 输出结果：

```
a: 2
o: 10
b: 5
```

D. 输出结果：

```
2: a
10: o
5: b
```

3. 对比 `std::vector<char> a` 和 `std::string a` ，下列哪种操作 `std::vector` 和 `std::string` 共同具有（不会造成编译错误）：

   A. `std::cin >> a`

   B. `a.append("abc")`

   C. `std::sort(a.begin(), a.end())`

   D. `a.length()`

多选题：

4. 在C++中，以下关于 `std::endl` 和 `'\n'` 的描述中，哪些是正确的?

   A. `std::endl` 会在输出流中插入换行符并刷新输出缓冲区，而 `'\n'` 只会在输出流中插入换行符。

   B. `'\n'` 会在输出流中插入换行符并刷新输出缓冲区，而 `std::endl` 只会在输出流中插入换行符。

   C. 相比于使用 `std::endl` ，使用 `'\n'` 可以减少清空缓存的次数，通常可以获得更高的效率。

   D. `'\n'` 是一个字符，而 `std::endl` 是一个函数。

5. 关于下列函数，说法正确的有（为简便省略了头文件）：

   A. 以下函数的功能是将 `target` 的前缀修改为 `prefix` ，但是算法复杂度可以进一步降低

```
void replace_prefix(const char * prefix, char * target){
    for (int i = 0; i < min(strlen(prefix), strlen(target)); ++i){
        target[i] = prefix[i];
    }
}
```

B. 以下函数的功能是将字符串数组 `suffixes` 中的字符串依次拼接到 `target` 后面，但是在不改变函数签名的情况下，可以通过优化内部实现，减少对象创建和拷贝等操作

```
void append_suffixes(string & target, vector<string> & suffixes){
    for(auto & suffix: suffixes){
        target = target + suffix;
    }
}
```

C. 以下函数的功能是将字符串数组 `suffixes` 中的字符串依次拼接到 `target` 后面，其中 `target += suffix;` 相比 `target = target + suffix;` 更加高效

```
void append_suffixes(string & target, vector<string> & suffixes){
    for(auto & suffix: suffixes){
        target += suffix;
    }
}
```

D. 以下函数的功能是将字符串数组 `suffixes` 中的字符串依次拼接到 `target` 后面，其中 `target.append(suffix)` 相比 `target = target + suffix;` 更加高效

```
void append_suffixes(string & target, vector<string> & suffixes){
    for(auto & suffix: suffixes){
        target.append(suffix);
    }
}
```

6. 对于该类：

```
class Student {
public:
    std::string name;
    int score;

    Student(std::string n, int s) : name(n), score(s) {}

    bool operator<(const Student& other) const {
        return score < other.score;
    }
};
```

我们希望按照score从**高到低**排列，以下哪些选项是正确的?

A.

```
std::vector<Student> students;
// 省略加入元素的操作
std::sort(students.begin(), students.end());
```

B.

```
std::vector<Student> students;
// 省略加入元素的操作
std::sort(students.rbegin(), students.rend());
```

C.

```
std::vector<Student> students;
// 省略加入元素的操作
std::sort(students.begin(), students.end(), std::greater<Student>());
```

D.

```
std::vector<Student> students;
// 省略加入元素的操作
std::sort(students.begin(), students.end(), [](const Student& a, const
    return a.score > b.score;
});
```

7. 对于以下代码，正确的有

```
#include <memory>
#include <iostream>
using namespace std;
void f(shared_ptr<int> p1){
    cout << p1.use_count();
    static auto t2 = p1;
}
int main()
{
    int *x = new int(1);
    {
        shared_ptr<int> p1(x);
        f(p1);
        cout << p1.use_count();
        f(p1);
        cout << p1.use_count();
    }
    //(1)
    return 0;
}
```

A. 输出是 2232

B. 输出是 1222

C. 在(1)处， `x` 已经被析构

D. 若在(1)处加入 `shared_ptr<int> p2(x);` ，则可能导致 `x` 被析构两次。

8. 以下哪些选项是正确的：

A. `shared_ptr` 可以复制构造、移动构造，但 `weak_ptr` 和 `unique_ptr` 只能移动构造，不能复制构造 B. 当动态创建的对象被多处引用，可以使用 `shared_ptr` 自动的管理该对象的销毁 C. `std::function` 在编译期确定函数调用的地址 D. `std::function` 可

以存储任何可调用对象，例如函数指针和 lambda 表达式

语言和编译选项

#	名称	编译器	额外参数	代码长度限制
0	answer	cp		1048576 B

递交历史

#	状态	时间
表中没有数据		

递交答案

选择文件 未选择任何文件 添加

提交