那我直接给你一整套"大综合卷"，你就当正式机试/面试笔试来做。
不带答案，全部按题型分块，每题后都标注【考点：...】方便你对照复习。

C++ 第一章～第三章大综合试卷（面试风格）

> 以第三章为主线，滚动复习第 1～2 章。
> 建议：真的可以当正式模拟卷来写一遍，做完我们再一起对答案、总结八股话术。

一、概念 / 简答题

1. 顺序容器与关联容器选型（典型 EDA 场景）

某 STA 工具中有如下三类需求：

1. 保存一条 Timing Path 上的所有 `Node*`，要求**按拓扑顺序**遍历，偶尔在尾部追加节点。
2. 根据 `std::string` 类型的 net 名查 `Net*`，查找非常频繁，对顺序无要求。
3. 需要按 slack 从小到大输出所有 path 的统计结果，每次 path 统计完就插入结果，后面经常要从"小 slack 开始"遍历。

分别选择最合适的容器，并说明理由（至少从**有序/无序、随机访问能力、增删效率**三个方面做对比）。
【考点：3.2 顺序容器(vector/list/deque)、3.3 关联容器(map/unordered_map)、1.1 内存+复杂度直觉】

---

2. `std::vector` 的 capacity / size 与内存特性

解释以下概念并说明它们的关系：

1. `size()` 与 `capacity()` 的区别；
2. `reserve(n)`、`resize(n)`、`shrink_to_fit()` 分别做什么；
3. 为什么频繁往 `vector` 尾部 `push_back` 时，偶尔会非常慢？这和底层内存分配有何关系？

【考点：3.2 顺序容器-vector、1.1 基本内存概念】

---

3. `map` / `unordered_map` 在增删改查上的复杂度与有序性

对比 `std::map` 与 `std::unordered_map` 在以下方面的差异：

1. 插入 / 查找 / 删除 的平均 & 最坏时间复杂度；
2. 遍历时的 key 顺序；
3. 是否支持 `lower_bound` / `upper_bound` 以及常见适用场景。

【考点：3.3 关联容器(map/unordered_map)、3.4 lower_bound 概念】

---

4. 迭代器类型与常用算法

回答：

1. `std::sort` 对迭代器的要求是什么？为什么 `std::list` 不能直接用 `std::sort`？
2. `std::reverse`、`std::lower_bound` 分别对迭代器能力有什么要求？
3. 给出几个典型容器及其迭代器类型：`vector` / `list` / `map` / `unordered_map`。

【考点：3.4 常用算法 + 3.2、3.3 容器迭代器】

---

5. 指针 vs 引用在容器中的使用

分别讨论以下三种容器元素类型的区别和适用场景（可以结合 EDA 中 Node/Net 结构体来想）：

1. `std::vector<Node>`
2. `std::vector<Node*>`
3. `std::vector<std::reference_wrapper<Node>>`（可选扩展，没学过可以跳过）

从**对象生命周期、拷贝成本、所有权、接口设计**几个维度说明优缺点。

【考点：1.2/1.3 指针 & 引用、1.5 参数传递语义、3.2 顺序容器】

---

6. const 与容器：只读视图与可写视图

考虑如下接口：

```
class Graph {
public:
    const std::vector<Node*>& nodes() const;
    std::vector<Node*>&       nodes();

private:
    std::vector<Node*> nodes_;
};
```

1. 为什么需要同时提供 const / 非 const 两个重载版本?
2. 对以下代码，分别写出 `n1`、`n2` 的类型，并说明能否通过它修改 `nodes_` 中的指针：

```
Graph g;
const Graph cg;

auto& n1 = g.nodes();
auto& n2 = cg.nodes();
```

【考点：1.4 const、2 类（成员函数 const）、3.2 顺序容器 + 引用】

---

7. 参数传递：`const T&`、`T&`、`T`、`T&&` 在接口设计中的区别

对比以下函数参数风格（假设 `Path` 是一个较大的 class）：

```
void foo1(Path p);
void foo2(const Path& p);
void foo3(Path& p);
void foo4(Path&& p);
```

1. 分别说明拷贝开销、是否允许修改、典型使用场景（例如仅读、会修改调用者对象、用作移动语义等）；
2. 如果你要写一个“只读分析 Path，不修改原对象”的接口，以及一个“消耗 Path 内部资源，把结果移动出来”的接口，分别选哪个形式？说明理由。

【考点：1.5 参数传递语义、2 类（移动语义预热）、3.1 现代 C++ 语法（右值引用）】

二、代码阅读 / 输出题

8. `vector` 扩容与迭代器失效

阅读代码：

```
std::vector<int> v;
for (int i = 0; i < 10; ++i) {
    v.push_back(i);
}

auto it = v.begin(); // 指向 v[0]

for (int i = 0; i < 1000; ++i) {
    v.push_back(i);
}

int x = *it;
```

1. 这段代码是否有 UB（未定义行为）的风险？说明原因；
2. 如果我们在 push_back 前调用 `v.reserve(2000);`，对迭代器失效情况有何影响？

【考点：3.2 顺序容器-vector + 1.1 内存概念 + 迭代器失效】

---

9. `map` 的有序性 & 遍历顺序

```
std::map<int, std::string> mp;
mp[10] = "ten";
mp[3]  = "three";
mp[7]  = "seven";

for (const auto& kv : mp) {
    std::cout << kv.first << " ";
}
```

1. 实际输出顺序是什么？
2. 如果把 `std::map` 换成 `std::unordered_map`，输出顺序还可预测吗？简要说明原因。

【考点：3.3 关联容器(map/unordered_map)、3.1 范围 for + auto】

---

10. `operator[]` 在 map / unordered_map 中的插入语义

```cpp
std::unordered_map<std::string, int> mp;

mp["A"] = 10;
int x = mp["B"];
auto it = mp.find("C");

if (it == mp.end()) {
    mp["C"] += 5;
}
```

回答：

1. 执行完 `int x = mp["B"];` 后，`x` 的值是什么？此时 map 中有哪些 key，值分别是多少？
2. `find("C")` 是否会导致插入新元素？
3. `mp["C"] += 5;` 执行后 `mp["C"]` 为多少？
4. 总结 `operator[]` 与 `find` 在"是否会插入新 key"上的区别。

【考点：3.3 关联容器-unordered_map 的 operator[] / find 行为】

---

11. `std::sort` + lambda 排序 & 比较器逻辑

```cpp
struct Path {
    double slack;
    int    id;
};

std::vector<Path> paths = {
    {0.1, 3}, {0.05, 1}, {0.1, 2}
};

std::sort(paths.begin(), paths.end(),
          [](const Path& a, const Path& b) {
              if (a.slack != b.slack)
                  return a.slack < b.slack;
              return a.id < b.id;
          });

for (const auto& p : paths) {
```

```
    std::cout << p.slack << "," << p.id << " | ";
}
```

1. 写出最终的输出顺序（按 `slack,id` 对）；
2. 这个比较器是否满足"严格弱序"的要求？如果你是面试官，会不会追问？（不要求严格证明，只需说出直觉理由）。

【考点：3.1 现代 C++ 语法（lambda、auto）、3.2 vector、3.4 sort + 比较器】

---

12. `lower_bound` 的返回值含义

```
std::vector<int> v = {1, 3, 3, 5, 7};
auto it1 = std::lower_bound(v.begin(), v.end(), 3);
auto it2 = std::upper_bound(v.begin(), v.end(), 3);

std::cout << (it1 - v.begin()) << " " << (it2 - v.begin()) << "\n";
std::cout << *it1 << " " << *it2 << "\n";
```

1. 写出两行的输出结果；
2. 简要用"第一个 ≥ 目标、第一个 > 目标"的视角解释 `lower_bound` 和 `upper_bound`。

【考点：3.4 常用算法-lower_bound/upper_bound、3.2 顺序容器-vector】

---

13. `string_view` 生命周期与容器中的悬挂引用

```
std::string file_content = "clk net1 net2";

std::vector<std::string_view> names;
{
    std::string_view text(file_content);
    names.push_back(text.substr(0, 3));  // "clk"
    names.push_back(text.substr(4, 4));  // "net1"
}

file_content.clear();
file_content.shrink_to_fit();

// 下面访问 names[0], names[1] 是否安全？为什么？
```

请回答：

1. `names` 中两个 `string_view` 实际指向哪块内存？
2. `clear + shrink_to_fit` 后，这两个 `string_view` 是否仍然有效？
3. 如果你要在容器中长期保存这些名字，你会如何修改设计？

【考点：1.1 内存+生命周期、3.2 顺序容器-vector、3.1 现代 C++（string_view）】

三、设计 / 开放题

14. 设计一个 STA 结果存储结构

需求：你需要存储每个 endpoint 的 timing 结果：

- endpoint 名（string）
- worst slack（double）
- path 数量（int）
- 还希望以后能很方便按 slack 从小到大输出。

请设计一套合理的数据结构组合，并说明：

1. 使用哪些容器（例如 `unordered_map` + `vector` + `sort` 等）；
2. 每种操作的大致复杂度：
   - 插入/更新一个 endpoint 的结果
   - 查询某个 endpoint 的结果
   - 按 slack 从小到大遍历所有 endpoint

【考点：3.2/3.3 容器选型、3.4 sort、1.5 参数传递语义（接口设计时可以顺带思考）】

---

15. 类设计：Graph / Node 之间的关系与容器选择

简化版要求：

- `Graph` 管理一组 `Node`，每个 `Node` 有唯一 `id` 和 `name`；
- 需要：
  1. 按 `id` 遍历所有 `Node`；
  2. 根据 `name` 快速找到 `Node`；
  3. 保证 `Graph` 析构时不会内存泄漏。

请设计 `Graph` 的成员变量（只需要给出大概类型）并说明：

1. `Node` 放在 `vector<Node>` 还是 `vector<unique_ptr<Node>>` 还是别的形式？
2. `id->Node`、`name->Node` 分别使用哪种关联容器？
3. 如何避免重复存储 / 悬空指针 / 内存泄漏？

【考点：2 类设计、1.2 指针 & 引用、1.4 const、3.2/3.3 容器组合】

---

16. const 接口与非 const 接口的 API 设计

考虑如下类：

```
class Path {
public:
    const std::vector<Node*>& nodes() const;
    std::vector<Node*>&       nodes();

    double slack() const;
    void set_slack(double s);

private:
    std::vector<Node*> nodes_;
    double slack_;
};
```

1. 请解释为什么 `nodes()` 要做 const/非 const 重载，而 `slack()` 不需要这样写就够用了；
2. 如果你只有一个 `const Path& p`，还能不能修改 `p` 的 slack？为什么？
3. 在大型工程中滥用 `const_cast` 会带来什么问题？

【考点：1.4 const、2 类（封装）、3.2 vector】

---

17. 提高接口效率：值传递 vs 引用传递 vs 移动

原始接口：

```
void AddPath(std::vector<Path> paths);
```

它每次都会**拷贝一整个 vector**，成本很高。请给出两个更优的接口设计版本，并说明适用场景：

1. 一个适用于"调用方还要继续用 `paths`"的情况；
2. 一个适用于"调用方只构造好 `paths`，交给你之后就不再使用"的情况。

【考点：1.5 参数传递语义、3.2 顺序容器-vector、3.1 现代 C++ (&&, std::move 可顺带考虑)】

四、改错 / 重构题

18. `std::sort` 比较器写错 & 不能 sort 容器

下面代码想要按出现次数从大到小输出 gate 名，存在多个问题：

```
std::unordered_map<std::string, int> gate_cnt;
// 已经填充好 gate_cnt

std::sort(gate_cnt.begin(), gate_cnt.end(),
          [](auto& a, auto& b) {
              return a.second > b.second;
          });

for (const auto& [name, cnt] : gate_cnt) {
    std::cout << name << " " << cnt << "\n";
}
```

1. 请说明这里为什么不能直接对 `unordered_map` 调用 `std::sort`；
2. 请给出一个完整的、可编译的改写方案，实现"按出现次数从大到小输出"的目标（可以使用 `vector` 中转）；
3. 指出你写的 lambda 参数是否需要 `const&`，为什么。

【考点：3.3 unordered_map、3.4 sort + lambda、迭代器种类】

---

19. 悬空指针：容器中保存局部对象地址

```
std::unordered_map<std::string, Node*> name2node;

void Build() {
    Node n1;
    Node* n2 = new Node();

    name2node["n1"] = &n1;
    name2node["n2"] = n2;
}
```

1. 指出这里的生命周期问题：`Build` 返回后两个指针各自状况如何?
2. 改写这段代码，避免悬空指针和内存泄漏（可以使用 `std::unique_ptr<Node>` 或其他你熟悉的安全方式）；
3. 说明 `name2node` 放在函数内 / 类内 / 全局静态区，对这个问题有没有本质影响?

【考点：1.1 内存与生命周期、1.2 指针、3.3 关联容器-unordered_map】

---

20. const 错误使用：改变 const 对象

```
class Foo {
public:
    int& data() const { return x_; }

private:
    int x_ = 0;
};

void test() {
    const Foo f;
    f.data() = 42;
}
```

1. 解释这段代码为什么在语义上是"危险的"；
2. 如何修改 `Foo` 的接口，保证 const 对象不能被外部修改内部状态?
3. 如果你确实需要"只读视图"，建议返回什么类型?

【考点：1.4 const（成员函数 const 正确用法）、2 类封装】

五、实现 / 编程题（伪代码级即可）

21. 使用 `std::map` 实现一个简单的"延迟查表"

有一个升序的 delay 表（电压 vs 延迟）：

```
// voltage -> delay
std::map<double, double> delay_table;
```

请实现一个函数：

```
double LookupDelay(const std::map<double, double>& table, double v);
```

要求：

1. 如果 `v` 正好等于某个 key，返回对应 delay；
2. 如果 `v` 在两个 key 之间，用线性插值：
   [
   delay = d1 + (d2 - d1) * (v - v1) / (v2 - v1)
   ]
3. 如果 `v` 小于最小 key 或大于最大 key，可以选择简单返回边界值（说明你的选择即可）。

提示：可以使用 `lower_bound` / `upper_bound`。

【考点：3.3 map、3.4 lower_bound、1.5 const 引用传参】

---

22. 使用 `std::vector` + `std::sort` + `std::unique` 做去重

请写出一段代码，实现对 `std::vector<std::string> nets` 做去重，不要求保持原本顺序，可以改变顺序，要求：

1. 最终 `nets` 中每个字符串只出现一次；
2. 时间复杂度尽量低（不使用 `set` / `unordered_set`，而是用 sort + unique）。

【考点：3.2 vector、3.4 sort + unique 惯用法】

---

23. 统计最频繁出现的 cell 类型

给定一个 `std::vector<std::string> cell_types`，里面是每个实例的 cell 类型名（例如 `"NAND2_X1"` 之类），请写代码：

1. 使用 `std::unordered_map` 统计每种 cell 类型出现的次数；
2. 输出出现次数最多的前 3 名及其次数（如果种类不足 3 种，就输出全部）；
3. 允许你用 `vector` + `sort` 来完成"取 Top 3"。

【考点：3.2 vector、3.3 unordered_map、3.4 sort + lambda】

---

24. 关联容器自定义 key：边的 key（可选稍难）

定义一个边的 key：

```
struct EdgeKey {
    int from;
    int to;
};
```

1. 让 `EdgeKey` 能作为 `std::map` 的 key，按 `(from, to)` 字典序排序；
2. 让 `EdgeKey` 能作为 `std::unordered_map` 的 key，自己写 `hash` 和 `==`；
3. 简要说明：`std::map` 依赖什么比较关系？`std::unordered_map` 依赖什么函数？

【考点：3.3 关联容器自定义 key + 1.4 const（操作符重载中常见）、2 类】

如果你愿意，下一步可以这样玩：

- 你先挑一类题（比如：**"代码阅读 / 输出题 + 改错题"**）全部做完，把你的答案发给我；
- 我按"面试官 + 带你整理八股"的口吻给你逐题点评，帮你把"知道"升级到"会说 + 会写 + 会扯项目场景"的程度。