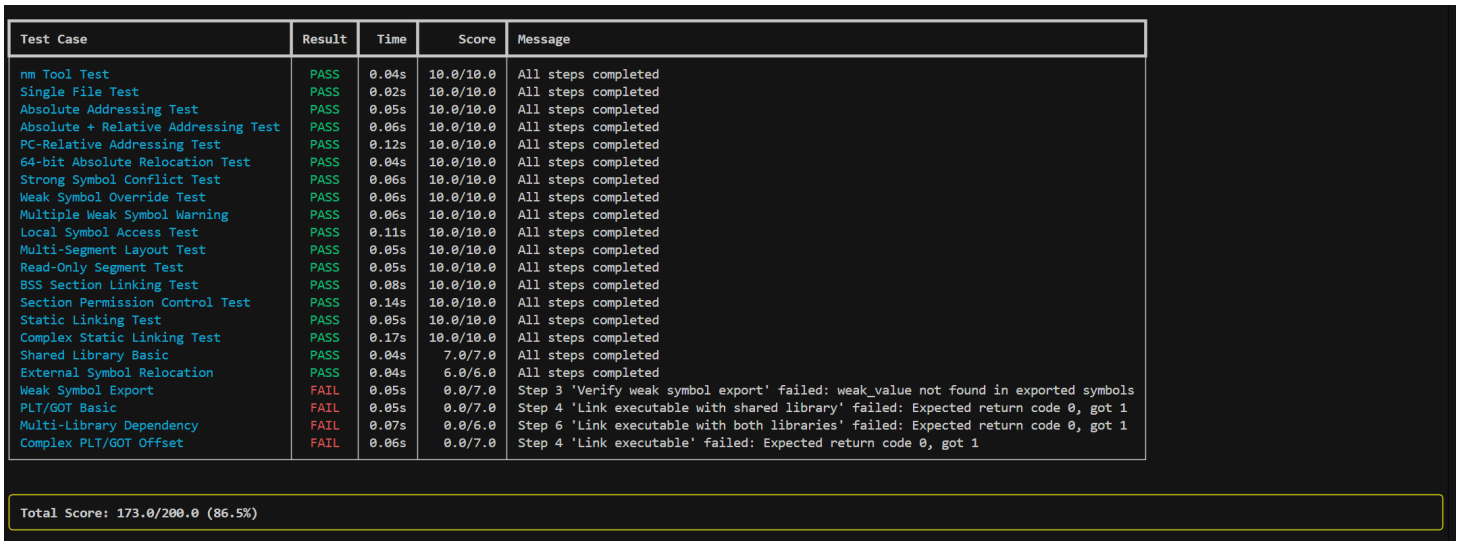
LinkLab 报告

姓名：朱照宇

学号：2024201532

Total Score: 173.0/200.0（最后四个测试点未能通过）



Test Case	Result	Time	Score	Message
nm Tool Test	PASS	0.04s	10.0/10.0	All steps completed
Single File Test	PASS	0.02s	10.0/10.0	All steps completed
Absolute Addressing Test	PASS	0.05s	10.0/10.0	All steps completed
Absolute + Relative Addressing Test	PASS	0.06s	10.0/10.0	All steps completed
PC-Relative Addressing Test	PASS	0.12s	10.0/10.0	All steps completed
64-bit Absolute Relocation Test	PASS	0.04s	10.0/10.0	All steps completed
Strong Symbol Conflict Test	PASS	0.06s	10.0/10.0	All steps completed
Weak Symbol Override Test	PASS	0.06s	10.0/10.0	All steps completed
Multiple Weak Symbol Warning	PASS	0.06s	10.0/10.0	All steps completed
Local Symbol Access Test	PASS	0.11s	10.0/10.0	All steps completed
Multi-Segment Layout Test	PASS	0.05s	10.0/10.0	All steps completed
Read-Only Segment Test	PASS	0.05s	10.0/10.0	All steps completed
BSS Section Linking Test	PASS	0.08s	10.0/10.0	All steps completed
Section Permission Control Test	PASS	0.14s	10.0/10.0	All steps completed
Static Linking Test	PASS	0.05s	10.0/10.0	All steps completed
Complex Static Linking Test	PASS	0.17s	10.0/10.0	All steps completed
Shared Library Basic	PASS	0.04s	7.0/7.0	All steps completed
External Symbol Relocation	PASS	0.04s	6.0/6.0	All steps completed
Weak Symbol Export	FAIL	0.05s	0.0/7.0	Step 3 'Verify weak symbol export' failed: weak_value not found in exported symbols
PLT/GOT Basic	FAIL	0.05s	0.0/7.0	Step 4 'Link executable with shared library' failed: Expected return code 0, got 1
Multi-Library Dependency	FAIL	0.07s	0.0/6.0	Step 6 'Link executable with both libraries' failed: Expected return code 0, got 1
Complex PLT/GOT Offset	FAIL	0.06s	0.0/7.0	Step 4 'Link executable' failed: Expected return code 0, got 1

Total Score: 173.0/200.0 (86.5%)

Part A: 思路简述

本链接器主要采用多次扫描的方法：

1. 第0次扫描：采用迭代算法，反复扫描库中的成员文件。只有当一个成员定义了当前“未解析符号集合”中的符号时，才将其提取并加入链接列表 `final_objects` 中。此过程持续直到无新成员加入，从而解决库内部的依赖及循环依赖问题。
2. 第1次扫描：遍历所有确定的输入文件，将同名节进行拼接。在此过程中计算每个输出段的虚拟地址，处理4KB页对齐，并记录下每个输入节在最终内存中的绝对起始地址 `input_section_addrs` ，为后续计算做准备。
3. 第2次扫描：遍历所有符号，结合Pass 1记录的节地址计算每个符号的最终绝对地址。根据强弱符号规则处理全局符号冲突，并将符号分为全局表 `global_symbols` 和局部表 `local_symbols` 。
4. 第3次扫描：遍历每个节的重定位项。对于内部已解析符号，根据重定位类型计算偏移量并直接修改机器码；对于共享库中的外部符号，则保留为动态重定位项，留待运行时加载器解决。

有两个重要的数据结构：

1. `SymbolInfo` :包含符号最终在虚拟内存中的绝对地址，符号类型，符号最终归属的输出段名称
2. `input_section_addrs` :它将 (输入对象指针, 节名称) 这一唯一标识映射到该节在最终可执行文件中的起始虚拟地址。这使得后续计算符号地址时，只需查表即可将相对偏移转化为绝对地址。

符号表：

1. `global_symbols` ： 类型为 `std::map<std::string, SymbolInfo>` ,以符号名为键，统一管理所有 GLOBAL 和 WEAK 符号。
2. `local_symbols` ： 类型为 `std::map<const FLEObject*, std::map<std::string, SymbolInfo>>` ,用两级映射。第一级以“输入对象文件的指针”为键，第二级以“符号名”为键。确保不同源文件中同名的 static 局部符号互不干扰

Part B: 具体实现分析

符号解析实现

1. **符号解析**采用分级隔离策略。局部符号存储在以文件对象指针为键的二级Map中，确保文件间同名static符号互不干扰；全局符号和弱符号汇入统一的全局符号表。针对 `bonus1` 的外部符号，在构建符号表时，代码显式跳过了未定义的符号，在处理重定位时，尝试在上述符号表中查找引用的符号名，若未找到，则将其视为外部符号，保留为动态重定位项。
2. **符号冲突**遵循任务四的文档要求：插入全局表时检查同名项，若两者均为 Strong，抛出 Multiple definition 错误；若现有 Weak 遇新 Strong，则覆盖为 Strong；其余情况不用处理。
3. **实现优化**上，利用 Pass 1 构建的 `input_section_addrs` 映射表，将地址计算简化为O(log N)的查表操作（**节基址+符号偏移**），极大提升了解析效率。并没有采用任务四的添加前缀名的方法来不同文件符号名干扰，因为这样会导致符号表变得复杂且难以管理（~~其实是因为那样做我有一个警告，有点难解决~~），直接在解析阶段用**二级映射**处理冲突更为简洁高效。
4. **错误处理**涵盖严格的边界检查：在链接可执行文件时，Pass 3会检测并报告所有未定义的强引用；解析时会校验符号所属节的有效性；对两个冲突的强符号会根据要求报错等。

重定位处理

1. **支持类型**覆盖任务三的所有要求，包括 `R_X86_64_32` 和 `R_X86_64_32S` 、 `R_X86_64_64` 以及 `R_X86_64_PC32` 。对于 `bonus2` ,未能实现 `R_X86_64_PLT32` 与 `R_X86_64_GOTPCREL` 类型，而且我发现 `fle.hpp` 文件中的 `RelocationType` 没有 `R_X86_64_PLT32` 类型,我实现的时候也不知道怎么处理，只有一个半成品，所以就放弃 `bonus2` 了。
2. **计算方法**基于Pass 2解析的符号地址S，Pass 1确定的重定位点地址P，以及保存在重定位项中的附加值A。绝对寻址直接计算S+A，相对寻址计算S+A−P。计算结果经位运算后，按小端序以及要求的位数逐字节填入输出节的对应偏移处。
3. **错误处理**实现边界检查，若计算出的写入位置超出节大小立即报错（如 `Relocation out of bounds` ）。同时区分模式：可执行文件遇到未定义符号即报错，而共享库则将其转换为动态重定位项，留待运行时解决。

段合并策略

1. **段合并**采用按序多路归并算法。我预定义了 `.text,.rodata,.data,.bss` 的标准输出顺序。外层循环遍历该顺序，内层扫描所有输入对象，利用辅助函数 `get_output_section_name` 进行前缀匹配（如将 .text.startup 归入 .text），将分散的输入节数据线性拼接至对应的输出缓冲区。
2. 在**内存布局**计算中，维护当前地址标志 current_addr（如果是EXE文件始于 0x400000，SO文件始于0）。每合并一个输入节，立即将其绝对起始地址记录至 input_section_addrs映射表，这一步是后续将相对偏移转化为绝对地址的关键（符号解析一步）。
3. **页对齐处理**：在每个输出段结束时，执行模运算 padding = (4096-addr%4096) %4096。对于非BSS段，向数据缓冲区尾部填充零字节；对于BSS段，仅调整地址游标。这确保了具有不同权限的段位于独立的物理页，满足系统安全要求。
4. **BSS 虚占用**：针对.bss 节，逻辑上仅累加 current_addr 以预留虚拟内存空间，但不执行数据拷贝，且不增加文件偏移量，从而生成紧凑的稀疏文件结构。

Part C: 关键难点解决

静态库的循环依赖与按需链接

- **难点描述**：静态库包含多个目标文件，标准要求仅链接程序实际需要的模块（即定义了当前未解析符号的模块）。难点在于库内部成员可能存在复杂的相互依赖甚至循环依赖（如A引用B，B又引用A），简单的单次线性扫描无法解决这种闭环，会导致符号未定义错误。
- **解决方案**:
  i. 维护两个集合： `defined_syms` （已定义）和 `undefined_syms` （当前缺失）。
  ii. 进入 `while(changed)` 循环，反复扫描库中的剩余成员。
  iii. 对于每个未提取的成员，检查其符号表。若它定义了一个存在于 `undefined_syms` 中的符号，则将其标记为提取（ `included=true` ,防止重复提取），将其新引用的符号加入 `undefined_syms` ，并置 `changed=true` 。
  iv. 循环直到某次完整扫描中没有新成员加入为止,此时 `changed=false` 。
- **方案效果**：成功解决静态库的循环依赖问题，确保了按需链接，避免了不必要的符号未定义错误。

外部符号与未定义引用的精准识别

- **难点描述**：链接器需要在重定位阶段区分“编程错误导致的未定义符号”和“指向共享库的合法外部引用”。由于符号解析是分阶段的，难点在于何时进行动态重定位项，准确断定一个符号的状态，并据此决定是报错、计算地址还是生成动态重定位信息。
- **解决方案**:
  i. 定义收集：构建全局符号表时，严格过滤掉 `type==UNDEFINED` 与 `section为空` 的符号，确保表中仅存定义的符号。

ii. 状态判定：在处理重定位时，首先尝试在全局表中查找符号名。
    - 若**Find**：判定为内部符号，直接获取其绝对地址进行静态重定位计算。
    - 若**Not Find**：判定为外部符号。此时根据输出类型分支：若是可执行文件（.exe），抛出Undefined symbol异常；若是共享库（.so），则将该重定位项原样保留，生成dyn_relocs。
- **方案效果**：精准识别外部符号，有效区分编程错误与合法外部引用，确保重定位阶段不会出现符号未定义错误。

Part D: 实验反馈

- **实验设计**：感觉必做部分 `task0-task7` 比较简单，但 `bonus1` 和 `bonus2` 真的很难，而且工作量很大，而且动态链接的过程经常破坏原来静态链接的部分（我一直报错），而且动态链接部分的 `debug` 太难了，我 `bonus1` 的最后一个测试点一直过不了（weak_value一直不在动态符号表中，但我又不知道错在哪），还有 `bonus2` 的 `R_X86_64_PLT32` 不在 `RelocationType` 结构体中，一涉及这个类型就报错，我根本不知道怎么处理，所以我就放弃了。
- **实验文档**：文档写的很好，很清晰，但感觉task0的时候一定要强调读一下fle.hpp，这回省去很多不熟悉fle系统而debug的时间；task2的文档可以再详细一点，毕竟感觉这个任务在task1的基础上是很大的跃升。还有task6的为什么需要对齐部分我觉得可以挪到task5中，因为task5的文档已经讲了节段了，大家在设计节段的时候知道这个机制可以避免 `mmap failed:Invaild arugement` 的报错（我在这个做task5报错的时候以为是重定位的问题，耽误了我好久，后来通过-v得到详细测试信息后问gemini才知道错误原因）；debug的指导教程真的可以写详细点，我一直不知道**代码中添加调试输出**怎么看,我试图让代码输出一些中间变量，但不知道怎么在终端看到，让我在做bonus1时很痛苦。
- **框架代码**：框架代码还是挺合理的，配合 `docs` 食用理解难度很低。（~~主要对框架代码也没什么研究~~）

参考资料 （可不填）

感觉 `docs` 讲的很清楚了，实现过程中遇到的问题都是问AI(~~因为找不到其它学校有这个lab~~)