1. C 课题

做一个文本编辑器，字符界面，或者图形界面。

做一个射击类游戏

电梯模拟器

玩某种牌类游戏（比如 21 点或者梭哈）

或者下某种棋（tic-tac-toe 不错）

管理一个小数据库，比如成绩单，可以输入、修改、保存、读取成绩单，并计算平均分

做一个小计算器，计算加减乘除，如果想提高难度，再加上括号。如果想降低难度，用“逆波兰式”。

1. 符号表

首先这已经超出 C 语言的范畴了，这是链接器、装载器的概念。

c 语言的程序可以分模块编译，然后链接在一起（静态链接）。有时候，一个模块需要引用另一个模块中定义的全局变量或者函数。但是，两个模块是分别编译的。编译器会遇到一个问题。以函数调用为例，如果模块 A.c 里的函数 f 调用了模块 B.c 里的函数 g，那么：编译器在编译 A.c 的时候，只知道 f 试图调用 g，但并不知道 g 在哪个模块里定义。

解决方案就是：
1. 编译 A.c 的时候，遇到调用 g 的地方，先把调用的目标空着，并在生成的目标文件（A.o）里写一个重定位记录（relocation entry）：“f 函数里 xxxx 位置试图调用 g，请在链接的时候把地址填上”，同时在 A.o 的符号表里写一条：“我需要 g，但不知道 g 在什么地方”。
2. 编译 B.c 的时候，发现有 g 的定义，就在生成的目标文件（B.o）里的“符号表”里放一个符号（symbol）：“g 在当前文件里的 yyyy 位置”。

然后，链接器拿到 A.o 和 B.o，发现：A.o 里有人需要 g 但不知道在哪里，B.o 里有 g。于是，就根据 A.o 里的那个重定位记录，把 A.o 里的代码修改一下，把 g 的地址填进去。

所以，这就是符号表：符号表里面包括很多符号（就是函数的名字和全局变量的名字等），以及每个符号的地址：可以是“已知地址”，也可以是“未知地址，需要从别的模块里解析”。

当然，现在的链接器还支持运行时的动态链接。这就需要符号表不仅仅要在编

译时保存留给链接器用，即使编译好的可执行文件（exe）和动态链接库（dll、so、dylib）也要保留符号表和重定位记录。这样，在运行一个 exe 的时候，装载器会把它依赖的 dll 也装载进来，然后通过符号表，把 exe/dll 之间互相的引用填好。然后程序就可以执行了。

【 在 kingsleynj 的大作中提到: 】
: 谢谢暖女神！
: 这应该就是常量折叠的意思吧？网上有帖子说 const 常量在 c 里面不能用来定义数组：

没有一点关系。“常量折叠”发生在编译阶段，而“符号表”是编译的产物。

: const int a = 5;

如果这个可以编译通过的话，可能和 C++11 放宽了数组大小的条件有关吧。C++对于编译器的要求比 C 更高。有可能新版 C++的编译器可以推断出“因为 a 是常量，所以可以用 a 的大小来定义数组长度”。

符号表里可能会有 a 这个项目，但存的是 a 的地址，目的是如果别的模块里想要访问这个全局变量 a 的值，链接器也要让它能够访问。

在 kingsleynj 的大作中提到: 】
: ?再次谢谢
: 可以再问一个问题吗？传引用和传指针到底有没有区别啊？引用和指针本身自然是有区别的，可是通过查看汇编代码，其实感觉传引用和传指针本身是没有区别的吧？传指针实现上还是和传值方式一样，会有一个临时变量的拷贝吧，引用呢？感觉看了很多博客，还是理不清二者的区别

本质上没有区别。

C++里，一个重要的概念是 object，不是“面向对象”的对象，是“存储空间”的意思。基本上，凡是有名字的东西都有存储空间。比如 int a;，定义一个存储空间，名字叫 a。但 42 就没有，它只是值。但如果 a=42，那么意思是把 42 存储到 a 对应的存储空间里。现在 a 的存储空间里储存着 42。

指针是对存储空间的引用（C++规格书的原话）。比如 int *p = &a;，那么 p 里面存着一个指针，它指向 a 的存储空间。

C++的表达式的值分为左值（l-value）和右值（r-value），其中左值是对应存储空间的，而右值就不对应。所以，a 是左值，而 42 是右值。至于 p，p 这个名字本身对应着存储空间，里面存着一个指针。所以 p 也是左值。表达式(*p)也是左值。意思是“p 指向的那个存储空间”。(*p)和 a 对应的存储空间是一样的。

因为指针也是值，所以，你可以把 p 的值传来传去。然后再用(*p)这样的表达式操作 a 的存储空间。这是指针的工作原理。

总结一下：C++里，用 int a;这种方法定义的标识符 a 对应存储空间，类型就是 int。存储空间里存储值。指针本身是一种值，它指向存储空间。(*p)这个表达式对应的存储空间就是 p 指向的存储空间。

然后再说 C++里的“引用”。

如果用 int &b = a;这种方式定义，那么 b 这个标识符的存储空间和 a 的存储空间是一样的。

就是这么简单。b 并没有自己的存储空间，可以认为 b 就是 a 这个空间的别名。

“引用”一旦定义，你就不能让 b 再去“指向”别的存储空间——在写下 int &b = a;的时候，b 的存储空间就固定了。

但是“指针”则不然。如果定义 int *p = &a;，那么 p 本身是个存储空间，里面存着一个指针。你可以再找另一个指针，存到 p 里。比如
int c; p = &c;
这样 p 就指向 c 的存储空间了。

所以，区别就是“指针本身是一个值，这个值执行一个存储空间；但引用只是另一个存储空间的别名而已，本身并不是单独的值”。

以上是指针和引用的语义。C++编译器只要能实现这个语义，想怎么实现都可以。比如，如果 int a; int &b=a;，b 和 a 在同一个函数里，编译器最直接的做法就是把所有提到 b 的地方换成 a。反正它们是相同的存储空间。