《编译技术》课程设计

申 优 文 档

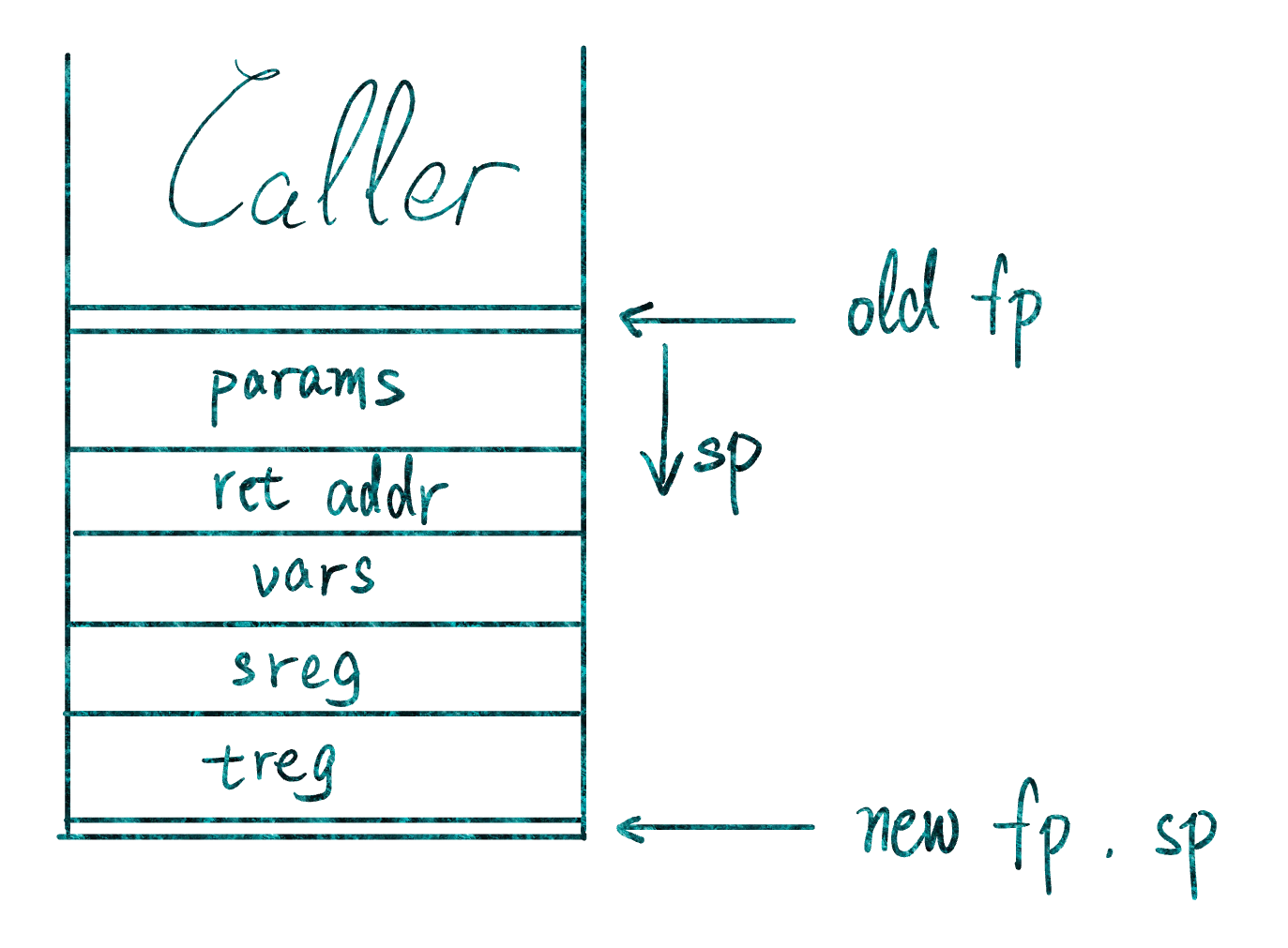
学号：\_\_\_\_\_\_\_15061200\_\_\_\_\_\_\_\_\_\_\_

姓名：\_\_\_\_\_\_\_李明轩\_\_\_\_\_\_\_\_\_\_\_\_\_

2019年 1 月 8 日

1. 一些设计细节
2. 运行栈的设计

运行栈的设计是我最满意的，在经历了两次小改之后迎来了现在这版满足目前所有需求的，且尽可能精简函数切换时保存寄存器数量的运行栈。简图如下，



运行栈设计的关键在于1.方便存取2.转场时少存东西3.保存足够的信息。第一点就是高低地址之争，听闻有的同学是从低地址向上增长，我觉得这在更大的工程中可能会引起不必要的麻烦，因为按照约定4GB内存空间里堆才是从低向高增长的，mars模拟器中的default模式也是如此，虽然我们没有堆也不会有问题出现，这样还方便我们思考，但如此略有违背内存空间的设计约定，私以为不是很好。

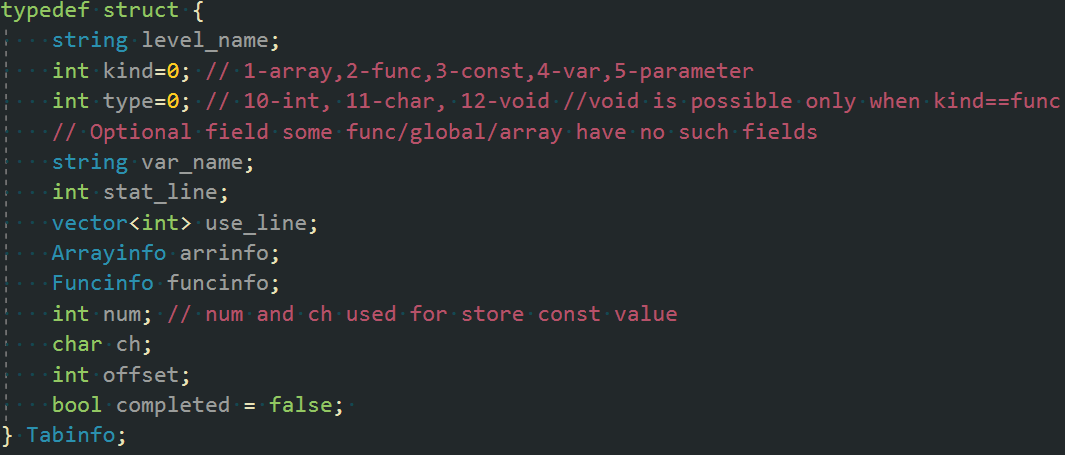
那么和理论课本上的运行栈相比较，我们的c0运行栈不需要display因为文法中只有全局和函数这两个层次，全局变量可以通过在mars中直接存取global区域得到。另外书上提到每次要存储fp, sp等信息，但细加思考就会发现，之所以要保存sp，fp等，是因为在动态分配存储空间时，我们不知道函数中的堆会占用多少空间，也就不知道sp会被挪动多少，所以需要有固定的位置来保存现场。但是在我们的文法中所有的变量所需空间都是确定的，因此可以采用静态空间分配，并进一步将sp，fp的保存简化为直接使用符号表信息增加或减少固定的量即可完成挪动sp，fp。继续深入，之所以先保存参数，再到函数变量和返回地址，是因为函数被调用时根据递归下降分析法，编译器会先识别到参数列表并计算参数，然后依次将参数push到栈上，最后才call func，因此参数会按顺序存放在我们的活动记录顶端。

最后则是保存寄存器的部分，这一块一开始我不明白为什么要区分caller saved和callee saved，后来随着理论课学习和实践，我才明白切换函数现场时临时寄存器池是默认全空的提供给callee的，而全局寄存器不一定全部都被使用到，所以是子函数负责随用随存。

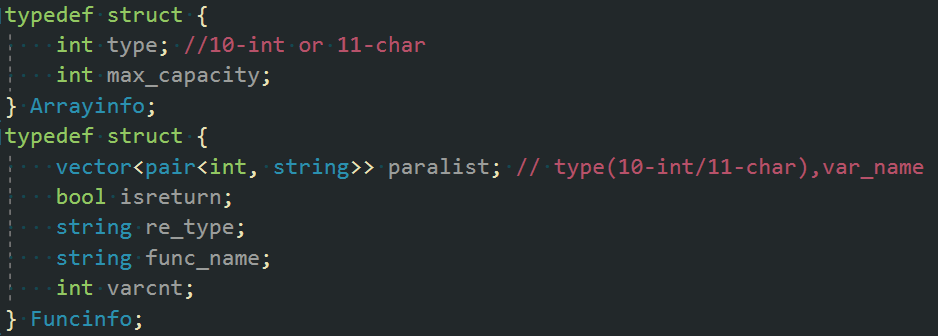
1. 符号表结构设计

符号表一开始设计的时候我曾想简单模仿课本上的栈式符号表，但后来考虑到c++丰富的STL支持和课设比较深入的需求，我决定使用一些特色数据结构重新设计符号表，最终结构如下

其中Tabinfo为自定义struct



其中Funcinfo和Arrayinfo均为自定义struct



总体来说，符号表设计中本人为了保证结构清晰和信息的完善，并考虑到可能的重名问题及查表便捷性的需求，将符号表设计为以上结构。最外层为hashmap因每个函数层次之间实际没有太多嵌套关系，本文法中只有两个层次(全局和各个函数包括main)。键即为该层次名字，值为该层次的符号表。而每个层次的符号表则是一个有序列表。这个列表的每个元素都由该标识符名称+标识符信息组成。每个层次列表的第一个元素都是该层次本身（也即该函数本身，全局则为虚设的global func代替）之后按照声明顺序依次插入列表。

深入到某个具体标识符的Tabinfo中来看，每个标识符都有一些必填项和根据类型功能决定的选填项。必填包括变量所在层次名（同前文函数名），其种类和类型。标识符名称，声明行，使用行等。数组信息和函数信息为可选的，而num及ch则是为了记录常量的数值，方便后续生成代码时直接查表替换常量。Offset是一个相当重要的变量，记录了该变量相对于函数内部所有非常量变量声明的偏移（包含数组元素个数，用于之后计算运行栈偏移）

进一步深入，数组信息Arrayinfo包含了数组类型和最大容量；函数信息Funcinfo包含了参数列表，是否返回，返回类型，函数名及包含的变量数量（接下来计算运行栈偏移会用到）。

我认为利用好STL设计一个**符合直觉**的符号表相当重要，这样在后续写代码和符号表交互的时候一则方便根据需求增删表项，二则方便自己思考构建逻辑。比如在我的设计中就是函数名作为层次名，每个函数统领一个自己的小表，每个小表内部按照顺序存储各个变量。（值得一提的是对于不同类型的量我的符号表可选表项部分也不同，这点统一存储统一查询的设计后期极大地方便了我的工程实现）

1. 从四元式变量到mips内存和寄存器

终于到了生成目标代码的阶段，这块最难的部分莫过于将变量名称和临时变量定位到具体的内存空间和寄存器中去，genMips\_findvar()函数是我写的最久，最复杂的一个函数，也是我debug最久，到现在也还不完全满意的一个函数。它主要负责了四个功能，分别是从栈到寄存器的读取和从寄存器到栈的读取。给定一个名称，我需要确认这个名称所代表的变量目前在哪里，是否计算出来了，是否已经存在某个寄存器中，是否在栈上，同时我的目标寄存器是否是一个特殊的寄存器（如a0 v0这样我必须生成指令把值写入，否则可以直接返回存储有被请求数值的寄存器编号），或者我需要计算这个名称和下标所代表的运行栈偏移量等等。这个函数承载了所有生成目标代码期间内存空间的定位工作，因此其结构设计和代码质量尤为重要。

个人在此处的感受是一定要**不嫌麻烦地深入到每一个子情况**细细考虑，尽可能用韦恩图帮助思考，覆盖到每一种可能的情况（为了防止自己review困难，我写了很多注释，甚至省略掉的情况也会写一个空的if else放在注释中，**正常运行时不会发生的情况也会写成else**，这一点习惯在后来极大地方便了我debug）。先把情况都考虑到，然后进一步慢慢优化代码结构，精简代码等等，切忌一开始便妄图设计一个完美优雅的函数体。

1. 参数寄存器的使用和运算顺序

这块的重要性在于函数的前几个参数可以直接无脑放在参数寄存器中，全程不用取出来，减少访存。唯一的问题在于多个函数嵌套调用时大家都会用到参数寄存器，怎么保证func2作为func1参数的时候，已经压栈压了一半的func1现场能够正确保存恢复呢？对于此我设计了一个栈式的函数压栈记录表，每一个函数压栈都会被记录当前进度，压栈完成函数成功调用才弹出栈顶记录，而一切相关的参数寄存器偏移量都根据这个记录计算得出。

此处需要注意的一点是压栈记录使用的关键字不能是函数名，而应该设计为每次函数调用的一个独立的编码（考虑到函数自身的嵌套调用），否则会导致记录混乱。

1. 最后的常数替换和（伪）启发式优化

由于时间关系，最后我实现的优化只有窥孔中的常数替换和常数表达式计算，以及（伪）启发式优化（史老师曾说编译器的优化最后是针对测试集合的优化，启发我最后实现出来这个结合了“人工智能”的自定义优化）。

常数替换和常数表达式计算都可以在生成代码阶段和四元式生成阶段直接进行。比如只要扫描到常量就从符号表中直接把数值取出来，替换进四元式中即可，而常数表达式计算可以放在代码生成阶段，如果某条四元式的两个操作数都是常量，那么就可以直接计算，然后将结果直接存储进预先分配的结果寄存器中即可。

重点说一下我的自定义优化。考虑到测试程序都具有一定的命名风格和变量使用特色，故而我们可以设定一系列规则，将函数中的多个变量按照规则排序然后依次分配进固定的一些寄存器中。在我的实现中，因为是面向测试程序的优化，所以可以更进一步，代码中设计好需要放入寄存器的函数和变量列表，之后根据测试程序代码，直接用“人工智能”识别出需要保存入寄存器的变量的优先顺序，然后写入代码进行生成即可。

1. 工程实现上的经验和思考
2. 工期安排和时间规划

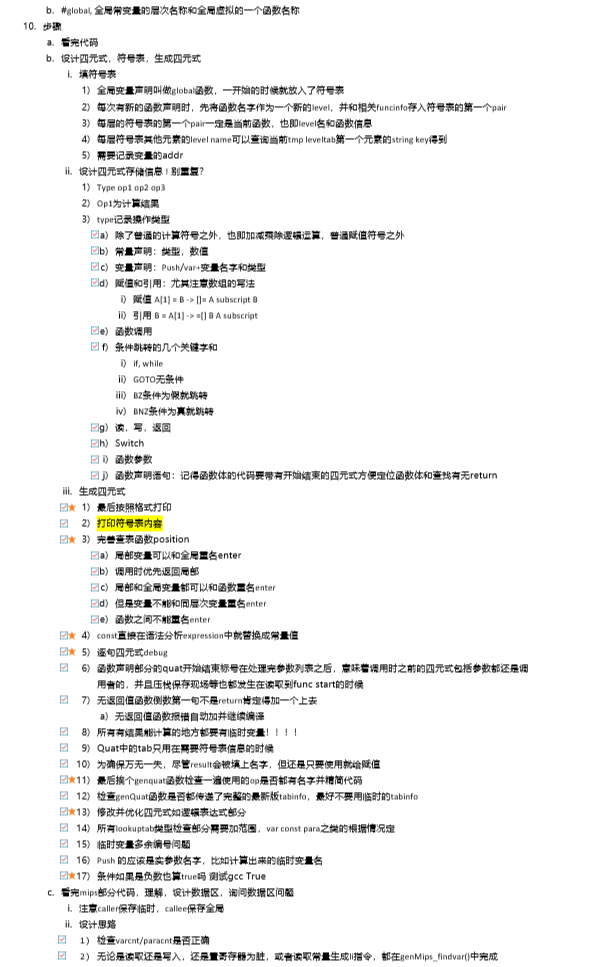
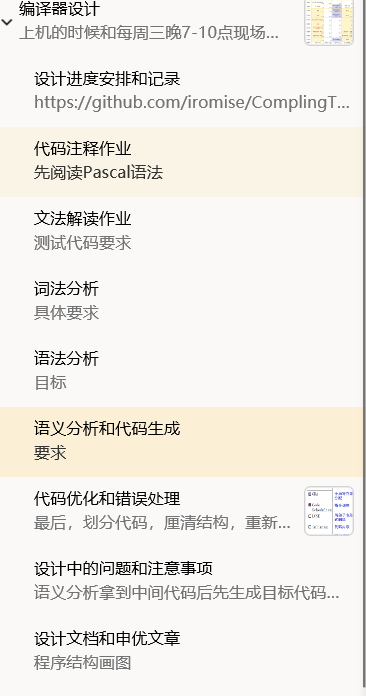
编译器课设是一个巨大的任务，所以分割和阶段安排显得尤为重要，这一点课程组已经帮我们做好了，在跟随课程组大进度的同时，我认为我们自己个人还需要将任务划分得更加细致一些，设立一个个人的任务清单/TodoList然后有条不紊地挨个实现完成。通过将任务细致地划分，可以最大程度避免最后关头“极限操作”或者突发情况导致的工期延后。

1. 代码质量和代码量

无论是和同级去年修过编译的同学还是在和学弟们交流中，大家都不可避免地提到代码量的问题，然后开始一番攀比，当然最后那些代码量少的人会受到大家的吹捧和艳羡的称赞。包括我自己也曾一度沉迷代码量，希望尽可能写出优雅简洁的代码来。但现在我认为，追求更高的代码质量和少的优雅的代码量二者是不等价的，有时候代码质量必须由复杂的细致的分情况讨论来保证，而以我们现阶段的水平，强行追求少的代码大部分时候只会弄巧成拙。

当然了，在完成某个阶段性任务后，我依然强烈建议大家回头看看自己的代码，争取优化自己之前写好的东西，这也是一次很好的学习机会。我也是用这种不断回头的优化方式，将整个编译器工程代码量控制在了3000行以内（包含了非常详尽的注释和格式等等），相比较部分同学5k的代码量，这一点让我非常有成就感。

1. 好的笔记和任务清单

考虑到任务量和绵长的工期，我们不可避免地会在赶工过程中注意到很多细节的同时又忘掉很多东西，并且思维的火花也是每天闪现又消逝，为了尽可能地提升我们的实现效率并方便我们思考，我认为保存一份自己的电子笔记是很重要的。以下是我的两张笔记截图：  


无论是自己进一步拆解的任务安排，灵光一现的想法，偶然发现的bug，论坛看到的好帖子，理论课后总结的感想，都可以记录在笔记本中，而且在记录自己思维过程的同时，就相当于和同学讲解了一遍自己的思考，也方便后续逻辑层面上的debug。这份笔记作为一个信息的汇集体和新想法的发源地，使我从中获益良多。

1. 杂谈随感

总的来说这个学期的编译原理课程设计让我受益良多，无论是理论和实践结合，还是工程能力培养，数据结构复习，C++强行入门，任何一个方面都足以称得上是计算机专业的最后一门大课。如何做好课设，我觉得首先就要夯实理论基础，很多时候自己闭门造车很久的东西其实都是错误或者低效的。这次的课设并不像以往数据结构或是某次算法作业题目一样，大家可以八仙过海各显神通，或者拍拍脑门查查资料就能解决问题，一切东西都需要建立在我们完全并且深入地理解编译原理细节的基础上，才能实现出一个优秀的编译器。

另外心态的调整也颇为重要，课程组安排的代码生成预提交满分，一二次测试补测机会这些机制都是给大家一个缓冲，同时严格要求的时间安排和工期也是帮助我们划分任务阶段，避免大家像当年的操作系统一样狼狈（听说有一些同学是在最后一周多补了两三个os实验）。虽然每周都要赶ddl是很痛苦的事情，但一个学期下来严谨的时间安排和自律也深深烙进了我们脑海中变成一种习惯。出于这一点，我们这个学期所经历的所有苦难，承受的所有压力，都终有回报。而前两次测试以及平时阶段作业提交中暂时的落后或者没过，只是一个小插曲，作为两次补交，测试一没过，第二次测试一二一起通过的我来说，深有体会。越是看到其他人都貌似领先于你的时候，越需要放平心态，用成长的眼光去看待自己暂时的失利，想着这又是一次学习改进的机会而已，然后努力去在下一次迎头赶上即可。

虽然实现过程中间有很多波折，比如论坛中细碎的约定更新，临近考核时突然添加的小要求，以及一些很激动的同学号召大家提意见等等，这些都像极了一个真实工程项目中可能出现的各种情况，是会有心情糟糕的时候，但也不是解决不了的问题，现在见识的这些我们以为的艰难困苦，都还远远及不上真实工作场景中的困难和窘境。总的来说，耐住性子，少说多做，时间会证明一切的。