《编译技术》课程设计申优文档

14005019\_胡杨

通过一个学期的编译技术课程设计，我实现了一个以扩充C0文法为输入语言的采用面向过程思想设计的编译器，可以生成mips汇编代码且在mars上模拟执行。在设计编译器的各个阶段，都遇到了不同程度难度的障碍与难题，将在这篇文章之中进行记录与分享。

一．词法分析

在我的词法分析程序中采用一符一类的的方法表示单词，这部分程序编写比较简单，就是一个自动机，不断地按照规则读入字符，然后进行拼接组成一个字符串小元素。

具体如下：

#define MAINTK 1

#define INTTK 2

#define CHARTK 3

#define CONSTTK 4

#define VOIDTK 5

#define IFTK 6

#define ELSETK 7

#define SWITCHTK 8

#define CASETK 9

#define DEFAULTTK 10

#define FORTK 11

#define PRINTFTK 12

#define SCANFTK 13

#define RETURNTK 14

#define ASSIGNSY 15

#define PLUSSY 16

#define SUBSY 17

#define MULTSY 18

#define DIVSY 19

#define LESSY 20

#define LESEQSY 21

#define GRTSY 22

#define GRTEQSY 23

#define NEQUALSY 24

#define EQUALSY 25

#define LSBRASY 26

#define RSBRASY 27

#define LMBRASY 28

#define RMBRASY 29

#define LBBRASY 30

#define RBBRASY 31

#define COMMASY 32

#define COLONSY 33

#define SEMICOLONSY 34

#define NUMET 35

#define CHARET 36

#define STRINGET 37

#define IDEN 38

#define CINTET 39

#define CCHARET 40

二．语法分析

在语法分析中我们不断调用词法分析的getsym()方法，就可以返回一个类型的int值，这就能告诉我们当前读入的Token类别，可以认为我们后面打交道的其实就是一堆int值而不是一个大字符串，这样就使源代码初步成为我们程序可以识别的小元素。

语法分析是一个很重要的部分，它关系到我们能否对源程序进行有效的分析，是我们语义分析的基础，这部分我们采用的是递归子程序法，让我们可以进行手工编程。

在语法分析方面，题目中的C0文法并无左递归问题，只有回溯问题。根据课本，为了消除回溯可以用改写文法或是预读字符的方法。课堂上老师着重讲了改写文法的方法，预读字符的方法只是介绍了一下。

顾名思义，递归子程序法，就要理解什么叫要递归、为什么要叫子程序，我的理解里，这个方法就是把语法元素与编程语言的一种对应，每一个语法元素就是一个子过程(或者说函数)，我们的程序根据词法分析的结果判断应该调用哪一条规则来进一步进行分析。

考虑到若改写成等价文法，使规则右部多个候选式首符号集不相交的话，虽然语法分析变得简单了，但改写后的规则中的符号含义变得不明确，本来是语义连贯的动作符号语义子程序，可能因为文法的改写被分到不同的语法递归子程序中，参数传递复杂而繁琐，而且模糊了语义编写时容易出错。反而给语义分析带来更大的麻烦。因此采用预读字符的方法处理回溯问题，由于在预读字符时只是进行语法分析，并没有语义处理，因此不会出现课本上所说的―相关的语义分析工作在回溯时也要推倒的情况，这两种方法在运行速度上并无本质差别。

三.符号表管理

在符号表管理这个部分，我采用了分表结构的组织管理，其基本思想是，每当编译程序扫描到一个分程序结构开始时，为该分程序建立一张符号表，在该分程序中定义的标识符，都被登录在该符号表中。而当编译程序扫描到一个分程序的结束时，编译程序释放为该分程序所建立的符号表。

编译程序扫描到某个分程序时，符号的登录是在为该分程序所建立的符号表中进行，而符号的查找是首先在该分程序符号表中进行；若没有查到，再根据分程序的层次结构，逐层向外地依次查找各层符号表。一直到查到为止。若所有符号表都已查完仍未找到，则表示有词法错误。在我的编译器中我使用了栈式符号表来存储，这样可以更好的适应作用域的嵌套，其实仔细观察我们文法也可以发现其实扩充C0文法不支持多层的作用域嵌套，在具体实现时也可以不用栈式符号表。栈式符号表更加能够符合我们程序的分析过程，也有利用我们的扩展，所以最后还是使用了栈式符号表。

四．四元式生成

中间代码是经过语义分析后得出的，使用中间代码的好处我认为主要有：便于优化、便于使程序各个部分的模块化、分步来降低难度，还有就是我们常说的可移植性。

我的四元式设计如下：

|  |  |
| --- | --- |
| 四元式 | 含义 |
| int , , , a | int a; |
| const,int,5 , a | const int a = 5; |
| char, , 30, a | char a[30]; |
| []= , a , i , t | a[i] = t; |
| geta, a , n , b | b = a[n] |
| call, f , , a | a = f() |
| call, f , , | f() |
| prt , a , b , symb | print("a", b) |
| scf , , , a | scanf(a) |
| jmp , , , label | jump to label |
| lab:, , , labx | set label |
| ret , , , (a) | return a / return |
| func,int, , f | start of function int f() |
| para,int, , a | f(int a, ...) |
| end , , , f | end of function f |

在四元式生成中最难的部分即在表达式，项，因子互相嵌套调用的地方，如何在合适的地方插入四元式生成相应正确的表达式需要反复琢磨思考。其次，作为翻译，肯定就要保证等价性，所以要保证翻译以后的中间代码语义上和源程序是一样的，这个可以先设计出一套，如果在语义分析发现还少的话可以继续添加，但应该在语义分析前就有一套比较完整的四元式设计，这样才可以写语义动作。

同时，一个优秀的四元式设计会为目标代码生成阶段省下不少力气。

五．代码生成以及存储分配方案

我认为这一部分是整个编译课程设计中最难的部分。在编写之初，我甚至都不明白存储分配需要设计的数据结构表与符号表的具体区别是什么。因为这一部分就需要更深入的涉及到编译器的后端—代码生成部分了。同时，需要将四元式按照编译的规则生成相应的目标代码。我们首先需要熟悉mips体系结构以及汇编语言，才能更好地完成这一部分代码。同时，在四元式设计中的诸多小细节在这里就发挥用场了，比如printf 一个表达式时，在四元式设计阶段记录下这是char还是int型的表达式在这里区别就是汇编系统调用的不同。这样的小细节还有很多，体现了四元式设计的重要性。

我们需要熟悉mips的体系结构，也需要明白sp和fp指针，桢指针，栈指针如何保存，改变，恢复，mips分为代码段和数据段，在栈顶需要存哪些数据，全局变量存放内存栈中何处，函数调用时参数值如何利用栈在函数间传递，这都是需要我们反复思考，推敲得出的。

我设计的内存分配如下：

|  |
| --- |
| 栈顶 |
| 全局变量区 |
| 主函数变量区 |
| 参数区  上一个函数的sp  上一级函数的fp  返回地址ra  本级函数变量区 |
| 参数区  上一个函数的sp  …… |

六．错误处理

错误分为语法错误和语义错误。语法错误即根据文法严格读下一个应该出现的符号。语言语义，就是这段代码实际的含义。编程语言的代码必须有绝对明确的含义，这样人们才能让程序做自己想做的事情，在语义不通的情况下需要及时报错。比如最简单的一行代码：a = 1; 它的语义是“将32位整型常量存储到变量a中”。首先我们对“1”有明确的定义，它是32位有符号整型字面量，这里“32位有符号整型”就是表达式“1”的类型。其次，这句话成为合法的编程语言，32位整型常量必须能够隐式转换为a的类型。假设a就是int型变量，那么这条语句就直接将1存储到a所在内存里。如果a是浮点数类型的，那么这句话就隐含着将整型常量1转换为浮点类型的步骤。在语义分析中，类型检查是贯穿始终的一个步骤。类型检查要做到判定每一个表达式的声明类型，判定每一个字段、形式参数、变量声明的类型，判断每一次赋值、传参数时，是否存在合法的隐式类型转换，判断一元和二元运算符左右两侧的类型是否合法（比如+不就不能在bool和int之间进行，但是我得到的问法只包含CHAR和INT这两个类型，所以暂时不用考虑这个问题）。

出错处理部分是我觉得最繁杂的一个部分了。因为在不同的地方出错，跳读的区间是不同的。所以需要慢慢磨。但是由于技术性较低，所以花了一些时间就搞定了，复杂但不难。

七．优化

我设计的编译器完成了两个优化：

1.基于DAG图的公共子表达式消除：

算法实现：遍历四元式，首先划分基本块，在基本块内部循环遍历，找出具有相同操作符（+,-,\*,/,=），相同左操作数，相同右操作数的操作符为+,-,\*,/,=的四元式，即为在DAG图中具有相同子树的节点。更改该基本块中之后出现相同子树的四元式即为删除公共子表达式。

生成DAG的范围选择是我们首要需要弄清除的一个问题，这里我们采用了比较保守的做法，我们对一段全是运算的四元式来生成DAG图，对于其他语句不在建立范围内。

2.基于引用计数的全局寄存器分配：

算法实现：首先遍历一个函数内部的所有四元式，对所有出现的变量进行引用计数，然后按照引用的次数进行从大到小排序，为前7个变量进行全局寄存器分配$s1-$s7并进行记录。在此之后，对这7个变量的相关操作不必在内存中进行存取，而直接操作寄存器即可。

八．感想

通过该课程设计，全面系统的理解了编译原理程序构造的一般原理和基本实现方法。把死板的课本知识变得生动有趣，激发了学习的积极性。把学过的计算机编译原理的知识强化，能够把课堂上学的知识通过自己设计的程序表示出来，加深了理论知识的理解。以前对与计算机操作系统的认识是模糊的，概念上的，现在通过自己动手做实验，从实践上认识了操作系统是如何处理命令的，如何协调计算机内部各个部件运行，对计算机编译原理的认识更加深刻。课程设计中程序比较复杂，在调试时应该仔细，在程序调试时，注意指针，将不必要的命令去除。

在这次课程设计中，我就是按照实验指导的思想来完成。加深了理解文件系统的内部功能及内部实现，培养实践动手能力和程序开发能力的目的。理解了该知识点以及学科之间的融合渗透本次课程设计程序部分是用c语言编写的，把《计算机操作系统》，《编译原理》，《算法分析与设计》《c语言》四门学科联系起来，把各个学科之间的知识融合起来，把各门课程的知识联系起来，对计算机整体的认识更加深刻。使我加深了对《计算机操作系统》，《编译原理》，《算法分析与设计》，《c语言》四门课程的认识。