编译设计文档

18373599 崔建彬

一、概述

本次设计文档将从词法分析、语法分析、错误处理、生成目标代码及优化的角度分析分析本次课程实验。最后进行心得体会总结。

二、词法分析：

1.词法分析任务：

写一个词法分析程序，从源程序中识别出单词，将输入的被编译源程序写为testfile.txt, 输出结果为output.txt。要求规则如下图。

单词名称	类别码	单词名称	类别码	单词名称	类别码	单词名称	类别码
标识符	IDENFR	else	ELSETK	-	MINU	=	ASSIGN
整形常量	INTCON	switch	SWITCHTK	*	MULT	;	SEMICN
字符常量	CHARCON	case	CASETK	/	DIV	,	COMMA
字符串	STRCON	default	DEFAULTTK	<	LSS	(	LPARENT
const	CONSTTK	while	WHILETK	<=	LEQ	)	RPARENT
int	INTTK	for	FORTK	>	GRE	[	LBRACK
char	CHARTK	scanf	SCANFTK	>=	GEQ	]	RBRACK
void	VOIDTK	printf	PRINTFTK	==	EQL	{	LBRACE
main	MAINTK	return	RETURNTK	!=	NEQ	}	RBRACE
if	IFTK	+	PLUS	:	COLON		

简单来说，就是根据指定的词法规则，对目标程序进行扫描，进行词法分析，得到每个词与之对应的属性，方便进行后续错误处理、语法分析等。

2. 思路分析：

先处理非字符串、字符常量、整型常量、标识符以外的单词。即保留字。保留字可分为两种，多字符保 留字/ 单字符保留字。对于多字符保留字，我们构建有限状态机来识别，对于单字符保留字，我们直接 使用HashMap来对应。在识别保留字符的时候，我们优先识别长字符的保留字，来保证遇到">="时，不 会识别为">" 。

对于整型常量，字符串，标识符，我们采取有限状态机的策略，来进行识别的同时判断出其是否符合规范。

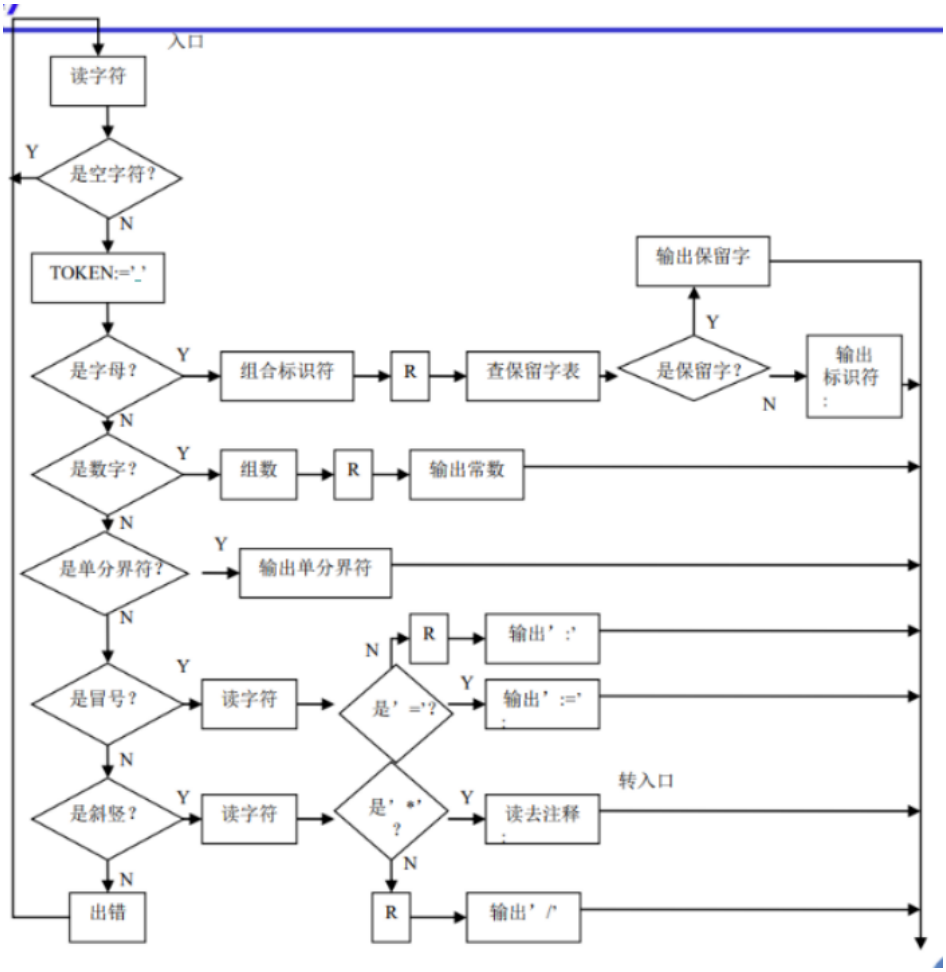
对于既是保留字又可能是标志符的单词，我们采取设定优先级的方法，优先实别保留字。

3.部分实现细节

a. 识别元素对应的文法：

1. <标识符> ::== 字母|<标识符>字母|<标识符> 数字
2. <无符号整数> ::== 数字| <无符号整数>数字
3. <单字符分节符> ::== ： |+|*|,|(|)|>|<|=|
4. <双字符分节符> ::== >=|<=|==|!=

b.设计构造



c.读入设计

一个字符一个字符进行读入，明显不能满足连续判断下一个字符的需求。所以我们先用stringstream将 内容从文件中一次全部读取出来，放到String里，再将string传入处理函数，来一个一个字符进行处理。

d.文件写入：

因为使用的是c++，所以可以方便的进行文件读写，先定义写入文件名和读入文件名，假设读入文件名 为readFile，写入文件名为writeFile，我们只需要，readFile>>String，即可读入一行，writeFile<<String，即可写入文件。

4.遇到的难点与bug:

1. 遇到的第一个bug应该是死循环，出现的原因是，假设了所有输入均为百分百正确，碰到其他字符如'.' 会卡住导致死循环。
2. 遇到的其他剩余bug，均为实现时不够小心，打错字导致输出类别码错误，或者对'==' ,'!='判断不好， 将其判断成为了'=' '='。

三、语法分析:

1.语法分析任务：

请根据给定的文法设计并实现语法分析程序，能基于上次作业的词法分析程序所识别出的单词，识别出各类语法成分。

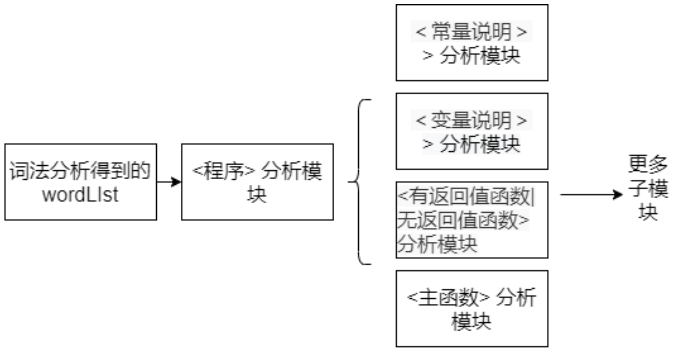
简单来说，就是根据词法分析的结果，对目标程序进行语法分析，并进行相应的处理。这样做，可以方便进行后续的错误处理、语义检查、生成中间代码等。

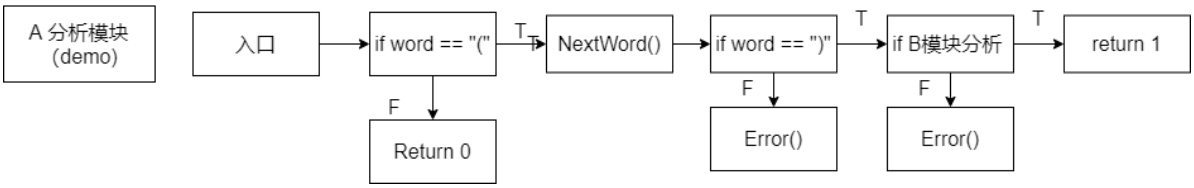
2.思路分析:

这次实验完成的是语法分析，根据指定的词法完成对输入程序的分析，并输出结果。整体来说，难道较词法分析有较大的提升。并不是语法分析难以处理，而是主要难点在于递归回溯的处理，error分析，和较大的代码量引起的可能的错误。

本次作业耗时较久，从构思到彻底AK，到重构再AK可能加起来用了18h左右，最后还是没有得到特别满意的代码架构，整个代码复用函数较少，导致整体较为冗长，希望后续能有提升。同时，这次作业还重构了上次词法分析一Main到底的代码，把他们变成了面向对象的设计结构，所以整体来说也是一个较大的改变。

3.架构设计与思路:





如图所示，对于每个模块，用if/else语句逐一判断，在不符合判断条件且与其他语句的FIRST()集合有交叉时候，就回溯，并返回0表示该模块判断失败。在其他情况下，若依然不符合判断条件，则直判定为错误。如果正确，模块就返回1，并记录该模块名字。

多个模块可能存在交替调用，自己调用自己的情况，所以需要认真理清楚每个模块，处理好回溯问题，防止错误调用。

4.遇到的难点与bug:

1. 在该分析程序失败时，应该回溯还是直接报错是一个比较难的选择。我最后的策略就是，看他与同级别的其他非终结符的FIRST集合有没有交集，但是至今为止我也不能100%保证代码完全符合这个规 则，所以后续我才用了更加保守的策略：即减少error的判断，在括号不匹配, 缺少其他标识符或符号时 候我才会判断error，更多的时候选择了回溯。
2. Bug 的出现主要是因为整体代码量过大，要写的分析模块过多，而且互相调用极多，这些都导致了我们在书写的过程中可能更容易出现bug，在找错的时候，更难以发现bug是什么导致的，只能选择单步 调试一步步走。这些无疑都让DEBUG的过程更加艰难，但是好消息是课程组依然提供了5组测试组并放 出了源码，这些更高效的帮助了我debug。（虽然我最后还是用了大约5h才完成debug任务）

四、错误处理：

1.错误处理任务:

请根据给定的文法设计并实现错误处理程序，能诊察出常见的语法和语义错误，进行错误局部化处理，并输出错误信息。为了方便自动评测，输入输出及处理要求如下：

（1）输入的被编译源文件统一命名为testfile.txt；错误信息输出到命名为error.txt的结果文件中；
（2）结果文件中包含如下两种信息：错误所在的行号 错误的类别码 （行号与类别码之间只有一个空格，类别码严格按照表格中的小写英文字母）

简单来说，就是根据语法分析，对程序的错误进行检测、输出、并进行一定程度的处理来方便其继续扫描后续程序。

2. 思路分析:

这次实验完成的部分是对程序的错处理，根据具体错误的地方输出错误的行号和错误的类型到Error.txt中。整体难度与上次作业相近，主要难在对语法分析部分的大规模修改，代码量大约增加了500-800行，但是其复杂度不比第三次语法分析作业低。

本次作业耗时两天，从从零开始写到debug到重构debug，整体来说还是比较耗时+费力的。面临的主要问题，就是在原有的代码上添加新的错误处理需求，以及构建符号表，和面对错误时如何让他修复来继续进行下一行代码检查上。

3. 设计细节:

1.对于缺少';' ,']', ')', 我们先用正常的if逻辑判断出来，如果发生缺失，采取回退一步的做法，再继续向下进行扫描。从而达到错误处理的效果。

2.对于非法符号或不符合词法的错误，我们直接在词法分析中进行判断，较为容易，对于错误处理，我们只需继续往下读取即可。即，忽略这个错误。

3.对名字重定义和未定义的名字，我采取符号表的方法来进行判断。因为本次作业处理不会如pascal一样支持多层嵌套，所以只需要建立两层符号表即可，即全局符号表和临时符号表。唯一比较难以处理的是，清空临时符号表的时机需要判断准确。

4.对于调用函数语句出现的错误，如参数类型不匹配，参数个数不匹配，我们仍然可以采用符号表的方法来处理，在函数定义时，加入其变量声明，类型，返回值类型，即可处理。

5.对于其他错误如数组初始化个数不匹配，缺少缺省语句等，只需要在声明数组/switch语句时进行判断即可。

6.除此之外，为了满足条件判断中出现不合法类型的判断等，我们还需要获取表达式的整体类型。在这里我让表达式如果声明成功，则返回其类型来帮助判断。

4. 遇到的难点及bug

难点主要在于在以前的代码上添加新的内容需求，以及许多想不到的细节。比如卡了我很久的：在函数声明的if语句中存在返回语句，这个返回语句有错误，我没有进行判断。比如return(a);中缺少;，我没有判断到。这些错误处理乍一看比较容易实现，而且比较基础，但是放到繁琐的递归下降法语法分析中时，就变得十分难以判断。因此，我de了很久的bug,最后还靠着同学们的帮助，才最终通过了错误处理。

五、代码生成

1. 代码生成任务：

在本次作业中，需要直接生成中间代码，并根据中间代码生成最终的目标代码同时输出到mips.txt中。本次作业分为了两次布置，在本报告中，将两次作业合并一起分析。

简单来说，就是根据语法分析，进行中间代码生成，根据中间代码生成目标代码(MIPS汇编指令)

2. 思路分析:

在刚看到题目的时候还是比较蒙的，阅读完2020官方参考资料(往届)后才有了整体的思路。就是先根据我们的语法分析的结果来生成相应的四元式，再在mips.cpp中根据四元式生成最终的目标代码。本次作业分为两次进行，每次耗时都大约在1-2天左右(从0写到debug通过)。目前还没有进行相应的优化，总代码大约增加了800行，但是其复杂度并没有太多的增加。

3. 设计细节:

a.四元式设计方案:

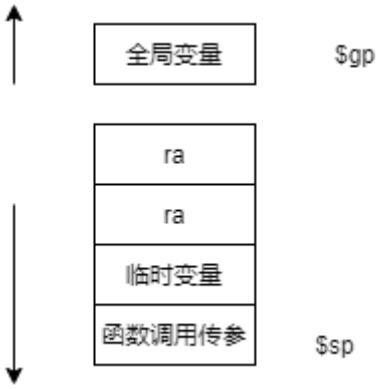
命名	操作	含义
GETARAAY	result = x[y1.1][y1.2]	获取数组，若为二元则将y分为两部分
PUTARRAY	result[x1.1][x1.2] =y	放入数组
ASSIGN	result = name,x=value	给变量赋值无y
PRITNF	result = value, x = value	type = 4:字符串, 5:下一行,else:表达式
SCANF	result = name	读入
MULT	result = x*y	乘法
MINUS	result = x-y	减法
BNZ	BNZ label(result) x= result	正确跳转 when result == 0
BZ	BZ label(result) x =result	不满足则跳转 when result ==0
LEQ	x <= y	<=
LSS	x < y	<
GEQ	x >= y	>=
GRE	x > y	>
NEQ	X != Y	!=
EQL	x == y	==
FUNC	result = name, x = type(void/int)	函数定义
RET	result = value	只有有返回值的函数才有
USE	result = value	调用函数时 传参用的
PARA	result = name, x = type	函数定义时的变量
RETVALUE	result = l临时变量名	调用有返回值的函数的值
SWITCH	result = exper，	声明是switch
CASE	result = label,x=exper	声明是CASE

b.四元式处理方案:

这些四元式存储在四元式数组中，在语法分析分析语法成分时，一并分析其产生的四元式。比如在分析<赋值语句>时，我们会根据其分析的目标，来生成ASSIGN四元式、PUTARRAY四元式等。在<条件判断>时，我们会生成IF、LEQ、LSS……等四元式。

得到生成的四元式表后，我们在mips.cpp中生成代码。我们根据不同的四元式来生成对应的目标代码。比如当读取到四元式SCANF时，我们需要先输出"li $v0 5"，"syscall",再通过符号表读取变量地址，生成mips"sw $v0 目标地址 "。正如上述操作，我们将四元式表中的四元式一一转换为目标代码。在未进行优化时，对变量的读取和写入均在内存中进行。

c.地址空间设计:



4.遇到的难点与bug:

本次作业的难点主要在于一开始的无从下手，直接从错误处理到生成目标代码有点不知所措。在大量阅读学长的报告后，我逐渐开始尝试完成作业。其次的难点就是在函数调用时，需要保存关键寄存器、并进行跳转。最后一些难点我认为在二维数组及条件跳转上。debug时，主要错误发生在表达式计算、表达式的传值上。除此之外，在条件跳转上也有一些错误发生。

六、优化部分:

1.优化问题分析:

优化主要是让生成的目标代码在目标机器上更快、更高效地运行。主要分为对中间代码的优化和针对生成代码的优化。本次实验中，因为实验原因，重点做了针对目标代码的优化。包括：全局寄存器的分配、临时寄存器的分配、常数合并、立即数的优化等。

2.具体实现细节:

a.全局寄存器分配:

本文中，全局寄存器分配采用简单的寄存器分配方式。即一个函数重置一次全局寄存器。在进入函数之前，用记数法统计出现次数最多的变量，并让他们由高到低排列，分别分配给S0-S8寄存器。在进入函数时，分别将这些寄存器保存在堆栈中，在进入下一个函数前，将这些寄存器清零。

相应的，我们要处理对变量的读取。当读取到相关变量时，先检查是否给这些变量分配了寄存器，若分配了寄存器，则直接从寄存器中读取，若没有，再读出其地址，从内存中寻找该变量。

b.临时寄存器分配

临时寄存器分配给临时变量。临时变量在做如下操作时可能会出现：a = b + 3 *d 那么我们的程序会将其处理为a = b+ @temp1 , @temp1 = 3 * d。 那临时变量就是指@temp1。

临时寄存器我们使用$t0-$t7给临时变量使用。这些变量有一个特点就是使用一次以后会销毁。所以我们只需要再读取到他们的时候为他们分配临时寄存器，在下次需要读取他们的时候，再将临时寄存器销毁即可。

c. 常数合并:

我们本次实验还做了常数合并。即面对a = 3 * 5 + 2时，我们会先把3* 5 + 2算好，直接用li语句赋值给a的寄存器。这样做大大减少了对alu相关指令的需求。

d. 立即数优化：

在正常的程序中，会出现大量的立即数(数字)。我们可以对他们进行优化。比如如果读到if(1 > 2)这种情况，我们可以直接在编译时比较1>2的大小并让其跳转，无需等到运行时才可以比较。同样的还有对数组下标的读取，我们可以直接在编译阶段读取其立即数计算出数组的位置，而无需在运行时再进行计算。同样的，我们还可以优化大量的加、减、乘、除法。降低ALU相关指令的数量和其他指令数量。

3. 优化部分总结：

优化是一个很有意思，很有成就感的过程。但是迫于时间紧张，我没有太多针对四元式的优化。因为mips是一个寄存器-寄存器指令集，所以合理地使用寄存器可以极大程度上的提高运行效率，寄存器的优化可以说是最重要的一个部分、也是见效最快的一个点。当然，还有许多其他优化可以做比如循环优化、多余代码删除等。但是很遗憾，这次没有能力和时间去完成了。不过总体来说，优化给我的过程是一个享受的过程，看着自己的程序一点点跑的更快，是一个非常有成就感的过程。

七、心得体会

编译课设是计算机学院的五大课设之一，难度较大，工程量较大，需要我们从零完成一个简单的编译器并进行一定程度上的优化。虽然费时较多，但整体上收获匪浅。

第一个收获就是更加了解c++了。因为本次实验的原因，让我解除了从未接触过的C++。从了解七vector可变长数组开始，到了解c++类的定义，再到进一步了解头文件与.cpp文件的关系，这些无疑都让我增进了对c++的认识。

第二个收获便是让我更加了解编译的过程。毫无疑问的是，从0写一个编译器是了解编译的最有效的过程。只有纸上谈兵是远远不够的，需要亲自实践才能有所收获。从构造自动机进行词法分析、到递归下降的语法分析、到复杂的错误处理及恢复现场、再到最后的生成中间代码和目标代码。这些无疑都是有一定难度的。但是在最后，我还是一一克服了这些难题，一次次进步。看到自己最终生成的目标代码在mars上运行时，内心无疑是十分激动的。

第三个收获便是增强了我的工程能力。完成一个编译器虽然也说不上很大的工程，但是代码量仍不低。一次次迭代、一次次文档、一次次实现新功能，这本身就是一种对工程能力的训练。完成编译器以后再回头看看大二时觉得困难的计组，突然也显得没那么困难了。

不过本次实验也有些许的不足和遗憾。那就是最后因为时间不够的原因，没有完成我能想到的全部优化内容，只做到了寄存器的分配，这一点还是稍有遗憾的。不过整体来说，本次课设还是十分难忘且让人充满成就感的。也希望后续课程组能越办越好，让更多的同学们享受搭建编译器这一过程。