華中科技大學

编译原理实验报告

学 生 姓 名: 胡思勖

学 号: U201514898

专 业: 计算机科学与技术

班 级: 计卓 1501

指 导 教 师: 徐丽萍

2018-06-23

目录

第 1 章　实验概述

1.1　源语言定义

本实验实现了一个 Decaf 语言的编译器。Decaf/Mind 是一种强类型的面向对象的、支持继承和封装的语言，此语言与 C/C++ 十分类似，但并不完全一样。本实验中实现的 Decaf 语言采用了 TAMU CS434 的 Decaf 语言定义[1]。

1.2　实验内容

本次实验实现的程序按照内容分为以下几个部分：

- 实验一：词法语法分析器的设计与实现：词法分析工具使用 Flex，语法分析工具使用 bison 完成。
- 实验二：语义分析以及符号表的设计与计算：生成抽象语法树，进行语义分析，设计符号表数据结构和关键管理功能。动态展现符号表变化过程。
- 实验三：中间代码生成：实现类型检查和控制语句目标地址计算，生成中间代码。
- 实验四：目标代码生成：在前三个实验的基础上实现 MIPS 平台目标代码生成。

1.3　实验环境

本次实验在如表1.1所示的环境下进行。

表 1.1: 实验环境

环境	版本
操作系统	Arch Linux 2018-06-10 更新
flex	2.6.4
bison	3.0.5
gcc	8.1.1
mips 模拟器	Mars 4.5

[1] https://parasol.tamu.edu/courses/decaf/students/decafOverview.pdf

第 2 章　词法分析与语法分析

2.1　实验设计

为了便于程序的设计，以及基于逻辑上的原因，在这一部分中首先编写用于语法分析的程序，然后根据此程序编写用于词法分析的程序。

首先，按照语法定义列出所有的终结符以及非终结符，如表2.1所示，然后定义运算符的优先级与结合性，如表2.2所示，最后按照语法规则完成 bison 的 EBNF 的编写。语法规则在1.1一节中已经给出。

表 2.1: 语法符号

符号	说明
tokenVoid tokenBool tokenInt tokenDouble tokenString tokenClass	终结符：变量类型关键字
tokenLessEqual tokenGreaterEqual tokenEqual tokenNotEqual tokenDims	终结符：二元运算符
tokenAnd tokenOr tokenNull tokenExtends tokenThis tokenInterface tokenImplements	终结符：逻辑运算符以及类相关运算符
tokenWhile tokenFor tokenIf tokenElse tokenReturn tokenBreak	终结符：流程控制关键字
tokenNew tokenNewArray tokenPrint tokenReadInteger tokenReadLine	终结符：内建函数关键字
tokenIdentifier tokenStringConstant tokenIntConstant tokenDoubleConstant tokenBoolConstant	终结符：常量
Constant Expr Call OptExpr LValue Type OptExt OptImpl ImpList ClassDecl Decl Field IntfDecl FnDecl FnHeader FieldList DeclList IntfList Variable VarDecl Formals FormalList VarDecls Actuals ExprList Stmt StmtBlock OptElse	非终结符

在进行 bison 语法分分析器的编写后，依照语法符号中的终结符给出词法的定义，并使用 Flex 编写词法分析器。其中，对于常量类型使用的正则表达式如表2.3所示。

表 2.2: 优先级与结合性

优先级	结合性	符号
高	左结合	`'='`
↓	左结合	tokenOr
	左结合	tokenAnd
	无	tokenEqual tokenNotEqual
	无	`'<'` `` `>' `` tokenLessEqual tokenGreaterEqual
	左结合	`'+' '-'`
	左结合	`'*' '/' '%'`
	无	tokenUnaryMinus `'!'`
	无	`'.' '['`
	无	tokenLower_Than_Else
低	无	tokenElse

表 2.3: 词法分析中的正则表达式

名称	正则表达式	说明
integerHex	`(0[Xx][0-9a-fA-F]+)`	16 进制整形
integer	`([0-9]+)`	10 进制整形
exponent	`([Ee][-+]?{integer\})`	科学记数法后缀
double	`({integer}"."[0-9]*{exponent}?)`	浮点型
stringBegin	`(\"[^"\n]*)`	字符串开始
string	`({stringBegin}\")`	字符串
identifier	`([a-zA-Z][a-zA-Z_0-9]*)`	标识符
operator	`([-+/*%=.,;!<>()[\]{}])`	运算符
commentBegin	`("/*")`	注释开始
commentEnd	`("*/")`	注释结束
commentSingleLine	`("//"[^\n]*)`	单行注释

在扫描到每个 token 后首先进行输出，然后直接返回 bison 中已经定义好的 token 类型即可。此外，词法分析器中还定义了三个额外的状态：COMMON、COPY 以及 COMMENTS，这些状态用于帮助处理 token 的扫描。其中 COPY 状态用于复制远吗中的内容，便于后续的调试以及错误处理，NORMAL 态为正常的词法分析状态，而 COMMENTS 态为程序处理注释时的状态，此时程序忽略其他所有的规则直到注释结束。程序初始时为 COPY 态，此后对于每一个 token 的扫描 COPY 态与 NORMAL 态交替出现，在复制源程序全部信息的同时完成对于词法的分析，输出以及传递给 bison 生成的程序。

2.2　实验过程

首先，按照实验设计中的语法规则编写 bison 以及 flex 程序（具体程序见附录B）。然后使用 bison 和 flex 处理得到 lex.yy.c 以及 y.tab.c。将生成的 c 程序进行编译和链接，得到可执行文件。此外，在使用 bison 的过程中会生成一个 y.output 文件，此文件为生成的语法分析器的状态机，可以利用这一文件以及 bison 的 interactive mode 对于状态机进行调试，以便于快速的检测出错误。

连接得到可执行文件后，编写一个 Decaf 程序以用于测试，此处编写的是一个排序 Decaf 程序，将测试程序输入到得到的编译器中，生成分析结果文件后观察文件内容。

2.3　实验结果及分析

输入测试程序 1（见附录B），得到的词法分析输出如图2.1所示（未显示完全），然后将生成的 Decaf 程序与源程序进行比对，发现没有缺失或错误翻译的情况存在，初步说明词法分析程序实现了计划的功能。

```
~/H/CompilerPrinciples  dev *…
./src/build/dcc samples/sort.decaf
Done.
~/H/CompilerPrinciples  dev *…  cat tokens.out
                int : tokenInt
                 [] : tokenDims
          ReadArray : tokenIdentifier
                  ( : tokenOperator
                  ) : tokenOperator
                  { : tokenOperator
                int : tokenInt
                  i : tokenIdentifier
                  ; : tokenOperator
                int : tokenInt
                num : tokenIdentifier
                  ; : tokenOperator
                int : tokenInt
                 [] : tokenDims
```

图 2.1: 词法分析输出

对于语法分析而言，由于实验进行到此处时只制定了语法规则，而未指定对应的动作，因此只能初步通过编译信息保证这一 bison 程序本身没有语法错误，而不能保证其功能正确。在接下来的章节中，将继续完善语法分析程序的动作，构建语法树并将其输出，以完整的检测语法分析器的正确性。

第 3 章　语义分析及符号表表计算

3.1　实验设计

这一部分主要完成对于语法分析工具的进一步完善，包括语法树的构建，语义分析，符号表的计算以及错误检查。前一章的内容为整个编译器的实现打下了基础，而这一章则是整个编译器的核心部分，这一部分所实现的程序让编译器真正理解其所编译的程序的语法含义。

3.1.1　语法树的构建

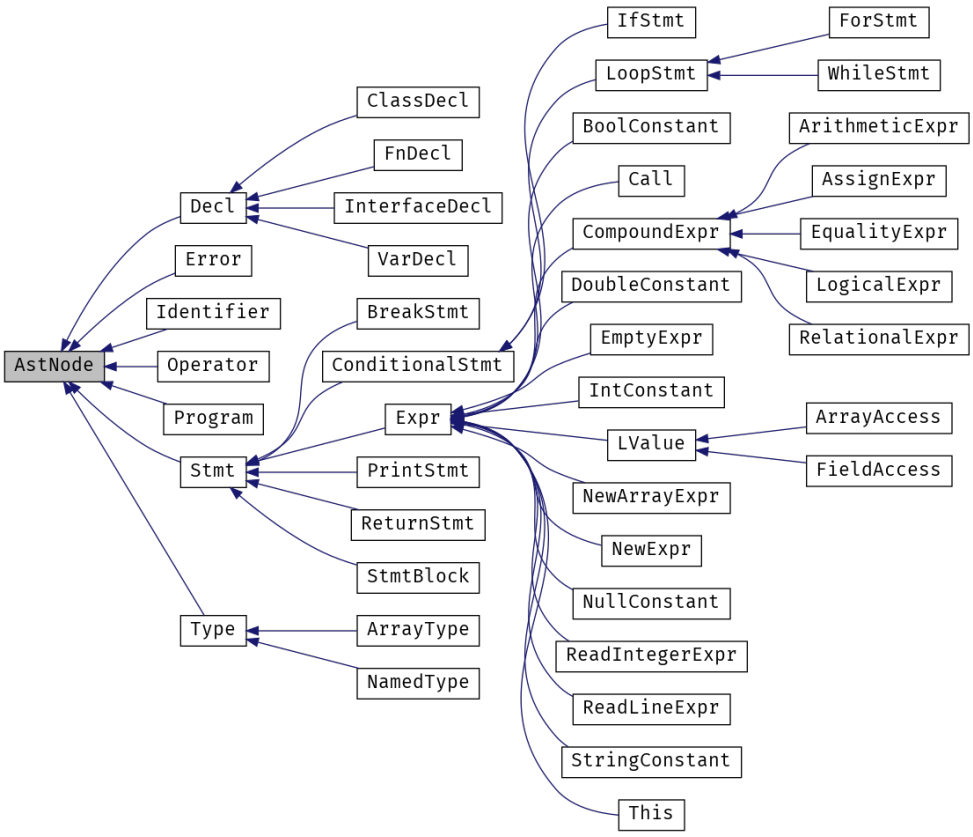

图 3.1: 语法树节点类型继承树

首先进行的是语法树的构建，而这其中第一步则是对于语法树数据结构的设计。利用 C++ 继承的

特性，可以轻易地将语法树不同节点的共同特征以及可以应用于这些特征的共同方法提取出来。在最终的设计中决定采用一个 AstNode 类作为语法树所有节点类的基类，然后对于语法树中可能出现的类型进行继承细分，最终，的代的所有类型节点的继承继承关系如图3.1所示。

对于基类 AstNode 而言，其包含了所有语法树节点的基本特征：用于调试和错误处理的 location 成员，表示这个节点对应成员在源程序中的位置，一个指向父节点的指针，一个用于表示作用域的 nodeScope 成员以及一些节点通用方法，如在节点作用域内查找声明等。最终，生成的语法树的每一条边都是双向的，而指向子节点的指针则存放于各个继承的类中，每个类的子节点类型以及数量不同。

为了在词法分析中能够正确的构建 AST，需要将构建语法树的代码写入 bison 程序中。此后，由 bison 生成的词法分析程序来构建语法树。

3.1.2　语义分析与错误处理

在语法树构建完成之后，接下来要进行的是静态语义分析。这一部保证了编写的语言在语义上是正确的。语法分析由节点类的 check 函数执行，每个节点的 check 函数负责检查自身是否符合语义限制。从根结点 Program 开始，第归地调用所有节点的 check 函数进行。check 检查的范围包括流控制检查，唯一性检查、上下文检查（包括使用声明的检查以及作用域的检查）以及类型检查。注意类型检查是对于 Expression（Expr 类及其子类）进行的，Expr 及其子类拥有 checkAndCompureResultType() 方法，通过 check 方法进行调用。这一方法不仅进行本节点的类型检查，还将本节点的类型计算并返回，用于上级表达式的类型检查。也就是说，类型计算在检查的过程中同时进行。

在检查发现程序由错误时，根据不同的错误调用相应的函数进行报错。为了方便错误类型的管理，所有的报错过程均存储于 ReportError 类中，作为其静态方法存在，所有的报错均是通过这个类进行的。目前，程序能国处理的错误类型如表3.1所示。注意部分错误由词法分析器或语法分析器抛出，这些错误不再静态语义分析的范围内，但为了便于管理，也将这些错误处理函数置于本类内。

表 3.1: 错误类型及错误处理函数

错误分类	错误分类	错误分类
词法分析器错误	UntermComment	注释未结束
	LongIdentifier	标识符过长
	UntermString	字符串未结束
	UnrecogChar	未知字符串
声明相关错误	DeclConflict	声明冲突（重复声明）
	OverrideMismatch	方法重载签名不符
	InterfaceNotImplemented	接口未实现
标识符相关错误	IdentifierNotDeclared	标识符未声明
表达式相关错误	IncompatibleOperand	运算符与又操作数不匹配
	IncompatibleOperands	运算符与双操作数不匹配
	ThisOutsideClassScope	类外使用 this
数组相关错误	BracketsOnNonArray	对于非数组取地址

错误分类	错误分类	错误分类
	SubscriptNotInteger	数组下标不是整型
	NewArraySizeNotInteger	使用 NewArray 时下标不是整型
函数/方法相关错误	NumArgsMismatch	参数数量不匹配
	ArgMismatch	参数类型不匹配
	PrintArgMismatch	使用 Print 时参数类型不匹配
域访问相关错误	FieldNotFoundInBase	为找到访问域
	InaccessibleField	访问权限错误（如访问内部类成员）
流控制相关错误	TestNotBoolean	测试语句返回类型非 bool
	ReturnMismatch	return 类型不匹配
	BreakOutsideLoop	在循环外部使用 break
链接错误	NoMainFound	主函数未找到

3.1.3　符号表构建

为了能够尽可能的减少遍历的次数，同时简化程序的逻辑，符号表的构建也在 check 中完成，并随着作用域分析（上下文相关性检查）的需要而惰性构建。

符号表使用 Hash 表实现，封装于 HashTable 类中，而此类为 stl 中 map 容器的简单封装。每一个 nodeScope 成员（表示一个作用域）持有一张符号表，特殊节点 Pogram 持有全局符号表。符号表将标识符映射到声明节点，这样在查找符号的时候就可以通过声明节点进一步的动态查询所需要的信息，同时减少了符号表的负担，使之不需要存储过多的信息。

在计算符号表时，由于符号表的计算时随着 check 函数而多心进行的，因此不需要第归进行计算，每次只需要计算本作用域的符号即可，随着 check 的进行而递归。对于每个 scope 所在的节点，采用 declareAll 注册其所有子节点的符号，并将其加入符号表中。为了方便调试，在注册符号的同时进行输出，而打印整个符号表。

3.2　实验过程

首先按照节点的继承树（图3.1）编写各个类型的节点的类定义。然后，将这些类加入语法分析其 parser.y 的 union 中，使得 bison 生成的程序能够将语法符号与类进行对应，从而执行相应的动作构建语法树。在编译与分分析器时，发现其不能正确的调用 yylex 进行词法分析。经过分析发现，由于在加入了语法树的构建后程序由 gcc 编译改为了 g++ 编译以构建 c++ 的类，从而导致 yylex 这一 c 函数不能正确被识别。在 yylex 的声明出加上`extern "C"`后，问题得以解决。

在修改完 parser 后能够正确的生成语法树了。然后修改每个节点的 check 函数以编写进行语义分析与检查的方法，值得注意的是，由于符号表计算随着语义分析中的上下文分析而惰性计算，但其他检查也可能用到符号表（如类型检查），因此符号表的计算必须在所有的检查前面完成。

3.3　实验结果及分析

在程序编写完成后输入测试用程序 1（见附录B），并对于输出进行分析。语法树构建的部分输出如图3.2所示。将其结构与源程序进行比较，其结构与程序的结构一致，说明语法树在结构上时正确的。

```
Program
├─ Function-Declaration : ReadArray
│  └─ Statement-Block
│     ├─ Variable-Declaration : i
│     ├─ Variable-Declaration : num
│     ├─ Variable-Declaration : arr
│     ├─ Variable-Declaration : numScores
│     ├─ Print-Statement
│     │  └─ String-Constant : "How many numbers to sort? "
│     ├─ Assign-Expression
│     │  ├─ Field-Acces : numScores
│     │  └─ ReadInteger-Expression
│     ├─ Assign-Expression
│     │  ├─ Field-Acces : arr
│     │  └─ NewArray-Expression : int
│     │     ├─ L-Value
│     │     │  └─ Field-Acces : numScores
│     ├─ Assign-Expression
│     │  ├─ Field-Acces : i
│     │  └─ Int-Constant : 0
│     ├─ While-Statement
│     │  ├─ Relation-Expression
│     │  │  ├─ L-Value
│     │  │  │  └─ Field-Acces : i
│     │  │  └─ Call : length
│     │  │     └─ L-Value
│     │  │        └─ Field-Acces : arr
│     │  └─ Statement-Block
│     │     ├─ Print-Statement
│     │     │  └─ String-Constant : "Enter next number: "
│     │     ├─ Assign-Expression
│     │     │  ├─ Field-Acces : num
│     │     │  └─ ReadInteger-Expression
│     │     ├─ Assign-Expression
│     │     │  ├─ Array-Acces
│     │     │  │  ├─ L-Value
│     │     │  │  │  └─ Field-Acces : arr
```

图 3.2: 构建的语法树（部分）

然后，对于语法分析与错误处理进行测试。对于每一种错误，均编写了一个简短的程序进行测试，由于篇幅冗长不在此处全部放出，此处选择其中的一个样例错误程序进行测试：

```
1  void main(){
2      int a;
3      a=b+c;
4      int b; int c;
5  }
```

可以明显的看出，b 和 c 在定义前被使用，此时程序应该汇报 baddecl 错误。程序的实际运行结果如图3.3所示，可以看出，程序正确的汇报了 baddecl 错误并指出了错误的发生地点。对其他错误的测试表明程序的行为预期的一致，可以说明在定的错误范围内程序均能正常工作，识别并汇报对应的错误。

```
~/H/CompilerPrinciples  dev *…
./src/build/dcc samples/baddecl.decaf

** Error line 4.
    int b;
    ^^^
** syntax error

Done.
~/H/CompilerPrinciples  dev *…                    Thu Jun 28 15:01:50 2018
```

图 3.3: 对于错误程序进行编译

最后，对符号表的输出进行观察，并与源程序进行比对，观察各个符号表所出现的作用域与作用域级别、类型是否与源程序中的一致。符号表以及作用域的输出如图3.4所示。经过与源程序的比对后，确认了所有的符号均被录入，所对应的作用域以及级别也与人工分析的结果一致，从而推断符号表的构建是正确的。

```
  ├─ [Program]
  (function) ReadArray
  (function) Sort
  (function) PrintArray
  (function) main
    ├─ [Function] ReadArray
    ├─ [Statement-Block]
    (Variable) i
    (Variable) num
    (Variable) arr
    (Variable) numScores
      ├─ [Statement-Block]
    ├─ [Function] Sort
    (Variable) arr
    ├─ [Statement-Block]
    (Variable) i
    (Variable) j
    (Variable) val
      ├─ [Statement-Block]
      │   ├─ [Statement-Block]
    ├─ [Function] PrintArray
    (Variable) arr
    ├─ [Statement-Block]
    (Variable) i
      ├─ [Statement-Block]
    ├─ [Function] main
    ├─ [Statement-Block]
    (Variable) arr
  └──└──└──└──
   0   1   2   3   ⟵ Scope level
```

图 3.4: 符号表与对应作用域输出

第 4 章　中间代码生成

4.1　实验设计

为了能够适应更多的平台，提高程序的可扩展性，进行与目标无关的优化，需要首先进行中间代码的生成。在本实验中，这一部在语法检查之后，将生成的抽象语法树转换为 TAC（三地址码）的形式。与语义分析一样，采用遍历语法树的方式完成这一步骤。

节点基类 AstNode 中定义了虚方法 generate 来完成这一步骤。由于不同的节点生成的 TAC 语句的数量、形式均有不同，因此在不同子节点类中重载这一方法并实现自己的 TAC 生成规则。由于这一部在语义检查后进行，所以在进行这一步骤时源程序已经可以保证处于无错误状态，不需要在进行额外的错误检查。

同样，TAC 生成从特殊节点，也就是根结点 Program 开始，递归的进行生成，对于每一个节点的子节点调用 generateAll 方法进行 TAC 的生成。为了更为方便的管理不同节点的生成规则，使用一个 CodeGenerator 类，此类中保存了所有不同类型节点的生成规则，而节点在调用 generateAll 方法时根据自己的类型以及自身生成 TAC 的需要调用 CodeGenerator 中的不同方法生成不同的 TAC 代码。

CodeGenerator 类本身持有一个线性表用于存放所有生成的 TAC 代码，这一线性表采用 stl 容器类 list 实现，其内存放的为 Instruction 类的实例。每一个 Instruction 实例表示一个 TAC 语句，而 Instruction 类本身包含其输出字符串，以及可能的三个操作数的地址。又由于不同的 TAC 语句有不同的格式，因此对于 Instruction 类进行继承，使其细分为如表4.1的拽同类别，便于进一步的格式化以及代码生成。

表 4.1: caption

Instruction 子类	**说明**
LoadConstant	读常量入临时变量
LoadStringConstant	读字符串常量入临时变量
LoadLabel	读标签地址入临时变量
Assign	赋值语句
Load	读内存语句
Store	写内存语句
BinaryOp	二元运算符语句
Label	生成标签

Instruction 子类	说明
Goto	无条件跳转语句
IfZ	为 0 跳转语句
BeginFunc	开始函数语句块
EndFunc	结束函数语句块
Return	返回语句块
PushParam	参数压栈语句
PopParams	参数出栈语句
LCall	远调用语句
ACall	绝对调用语句
VTable	为类生成方法虚表
DiscardValue	丢弃变量

在生成的过程中，临时变量和标号按照顺序进行编号，保证每个临时变量和标号的编号时全局唯一的，便于后续的目标代码生成。值得注意的是，表4.1中的 DiscardValue 并不实际生成一条 TAC 语句，只用于标明这是最后一次使用某一变量，随之将其丢弃；BeginFunc 和 EndFunc 则在生成函数语句块的前后被调用，用于生成调用函数的准备工作代码以及清理工作代码；函数调用采用调用者清理的方式，这一部由 CodeGenerator 中的 GenerateMethodCall 方法在生成函数调用代码前后调用 PushParam 以及 PopParams 的方式进行。

ast 中每个节点的生成 TAC 流程各不相同，以 for 循环（ForStmt 节点类）为例，其生成流程如表4.2所示。其余节点按照自己的需求组装 CodeGenerator 中的生成语句，此处不再全部给出。

表 4.2: ForStmt 生成 TAC 流程

#	函数调用	说明
1	init->generate(cg);	生成初始化语句
2	char *topLoop = cg->NewLabel();	注册循环体顶部标签
3	afterLoopLabel = cg->NewLabel();	注册循环体尾部标签
4	cg->genLabel(topLoop);	生成循环体顶部标签 TAC
5	test->generate(cg);	生成测试部分 TAC
6	cg->genIfZ(test->result, afterLoopLabel);	生成为 0 跳转 TAC
7	body->generate(cg);	递归生成循环体 TAC
8	step->generate(cg);	递归生成每次循环后执行代码 TAC
9	cg->genGoto(topLoop);	生成无条件跳转到顶部标签 TAC
10	cg->genLabel(afterLoopLabel);	生成尾部标签 TAC

4.2　实验过程

首先编写 CodeGenerator 类, Instruction 类以及 Instruction 类的所有子类, 为 TAC 代码的格式作出定义。为了便于调试, 在每个 Instruction 子类构造时将具体的 TAC 输出到文件中(除了 DiscardValue 不需要输出外)。然后将每个 Instruction 映射到 CodeGenerator 的不同生成函数中。生成函数的一般形式即为构造对应的 TAC 指令类并将其加入自己所持有的 code 列表中。在这一部完成后，程序已经具有生成所有类型 TAC 指令的能力。

然后, 对于每个 AstNode 的子节点, 重载 generate 函数, 按照自身类型的需求调用 CodeGenerator 的不同生成函数来生成 TAC 指令。在所有指令生成完成后, 完整的 TAC 程序应存放于 CodeGenerator 实例的 code 列表中。

4.3　实验结果及分析

程序编写完成后进行编译。同样，使用附录B中的 sort 例程进行测试。将其输入编译器中，并分析输出的 tac 指令。sort 程序输出的部分 TAC 指令如图4.1所示。由于 TAC 不能够运行，只能人工分析其正确性。经与源程序的比对，其逻辑与源程序基本一致，但正确性依然需要在后续的 MIPS 指令生成过后才能确认。

```
BeginFunc 128
_tmp0 = "How many numbers to sort? "
PushParam _tmp0
LCall _PrintString
PopParams 4
_tmp1 = LCall _ReadInteger
numScores = _tmp1
_tmp2 = 0
_tmp3 = numScores < _tmp2
IfZ _tmp3 Goto _L0
_tmp4 = "Decaf runtime error: Array size is ⩽ 0\n"
PushParam _tmp4
LCall _PrintString
PopParams 4
LCall _Halt

_tmp5 = 1
_tmp6 = _tmp5 + numScores
_tmp7 = 4
_tmp8 = _tmp6 * _tmp7
PushParam _tmp8
_tmp9 = LCall _Alloc
PopParams 4
*(_tmp9) = numScores
_tmp10 = _tmp9 + _tmp7
arr = _tmp10
_tmp11 = 0
i = _tmp11
```

图 4.1: sort 程序的 TAC 输出（部分）

第 5 章　目标代码生成

5.1　实验设计

在生成了 TAC 后，剩下的部分就是目标代码生成了。在这一部分中，程序将由能力生成可以运行的 MIPS 指令。

由于前面已经生成了三地址码，生成对应的 mips 指令只需将一种线性结构转换为另一种线性结构，这一部通过 CodeGenerator 类中的 generateFinalCode 完成。其中的核心部分在于寄存器的分配。可供分配的寄存器数量有限，而 TAC 中的临时变量一般远远超出可以分配的寄存器数量，因此需要知道那些变量不再使用，便于将其所绑定的寄存器空出给其他的变量使用。

要完成这一点，需要对程序进行数据流分析，而其中的第一步则是构建程序的流图。在本实验中，使用一个 ControlFlowGraph 作为存储流图的数据结构，对于流图这样一个有向无环图，ControlFlowGrap 采用双向邻接表的数据结构表示。对于个节点的出度使用一个矩阵，入度使用另一个，并分别建立 ForwardFlow 类以及 BackwardFlow 类作为原图的遮罩，便于图的前向遍历和方向遍历。

生成最终代码的大体框架分为 3 步：

1. 遍历 CodeGenerator 中的 TAC 指令，生成 CFG。
2. 遍历 CFG，生成活跃变量表。
3. 遍历 CodeGenerator 中的 TAC 指令，结合活跃变量表，完成最终的寄存器分配以及代码生成。

这些步骤均在 generateFinalCode 中完成。活跃变量表采用`list<map<Location,Instruction>>`的结构实现，其中，外层的 list 为源程序中不同的函数，每个函数有自己的 CFG, 而内层的 map 则是对于 cfg 进行分析后的活跃变量表，它将变量（Location 类，变量使用变量地址表示）映射到**最后一次使用它的指令**。这样，在后续的遍历过程中，只需要查找所使用的变量是否在活跃变量表中，如果在，则比对当前的指令是否与其最后一次使用的指令相同，如果相同，则说明这是最后一个使用这个变量的指令。此变量已经可以被丢弃，此后不会再被使用了。

第三步执行的正是这样的工作：遍历 TAC 指令并决定是否丢弃用到的变量。为了减少程序的耦合度，不因为丢弃变量而干扰其他类，使用上一章中提到的 DiscardValue 这一伪 TAC 指令生成目标指令，这样解绑寄存器的策略就决定与最终用于生成目标指令的类而不是 CodeGenerator 这一生成 TAC 指令的类了。

与从 AST 到 TAC 类似，为了便于程序的控制，减小各个模块之间的耦合度，采用 Mips 类进行最终的代码生成。由于产生的 TAC 指令与 LLVM 后端不兼容，要使用 LLVM 生成则过于复杂，因此使用此类直接硬编码生成 MIPS 指令。由于将 TAC 生成与最终代码的生成分离，因此可以轻易的将 Mips

类替换成其他类，用以生成 x86, arm 等一系列平台的目标代码。对于本实验中的 Mips 类而言，其通过不同的函数将 TAC 类映射到对应的 Mips 代码的生成方法，而 Instruction 类的各个子类通过虚函数重载了 generateSpecific 方法用于调用生成与自身 TAC 类相关的 Mips 指令，因此在 CodeGenerator 中只调用每个 TAC 指令的 generateSpecific 方法以及丢弃指令的 generateSpecific 即可生成最终的 Mips 代码。由于未作特别的代码优化，在 Mips 类中，除了特殊寄存器（如 sp）保留之外，其余的空闲寄存器均是随机分配的，以达到均匀使用所有寄存器的效果。

5.2　实验步骤

作为程序流图的基础，首先构建 ControlFlowGraph 类，然后对于其中的 mapLabels 以及 mapEdges 方法进行编写，对于普通 TAC 指令以及可能造成跳转的四种指令：LCall、ACall、IfZ 以及 Goto 进行映射，从而构建 CFG。

在 CFG 类实现完成后，实现 CodeGenerator 中的 generateFinalCode 方法，实现上述框架中的三个步骤，即生成 CFG、生成活跃变量表以及寄存器分配和生成最终代码。

最后，在 AST 的主节点生成 TAC 后调用 generateFinal 方法，构造出一个完整的编译器。

5.3　实验结果及分析

在编译器构建完成后，继续采用附录B中的 sort 程序进行测试。将程序输入编译器，得到的汇编文件部分如图5.1所示。

```
~/H/CompilerPrinciples  dev *…  cat a.out
                        # standard Decaf preamble
    .text
    .globl main
    jal main
    li $v0, 10
    syscall
_ReadArray:
                        #### BeginFunc 128
    subu $sp, $sp, 8    # decrement sp to make space to save ra, fp
    sw $fp, 8($sp)      # save fp
    sw $ra, 4($sp)      # save ra
    addiu $fp, $sp, 8   # set up new fp
    subu $sp, $sp, 128  # decrement sp to make space for locals/temp
s
                        #### _tmp0 = "How many numbers to sort? "
    .data               # create string constant marked with label
    _string1: .asciiz "How many numbers to sort? "
    .text
    la $t0, _string1    # load label
                        #### PushParam _tmp0
    subu $sp, $sp, 4    # decrement sp to make space for param
    sw $t0, 4($sp)      # copy param value to stack
                        # Last use of _tmp0. Discard register $t0
                        #### LCall _PrintString
    sw $t0, -24($fp)    # spill _tmp0 from $t0 to $fp-24
    jal _PrintString    # jump to function
```

图 5.1: 生成的汇编文件（部分）

由于之前在计算机组成原理课堂上实现的 Mips CPU 只能支持部分指令且不支持 Ascii I/O，因此最终选择在 Mips 模拟平台 Mars 上进行模拟。模拟结果如图5.2所示。测试中，对于 3, 6, 4, 2, 7, 12, 1, 0, 5, 10 这 10 个数进行排序。从图中可以看出，排序结果正确，可以说明 Decaf 程序正确的完成了它的功能，编译器功能正常。

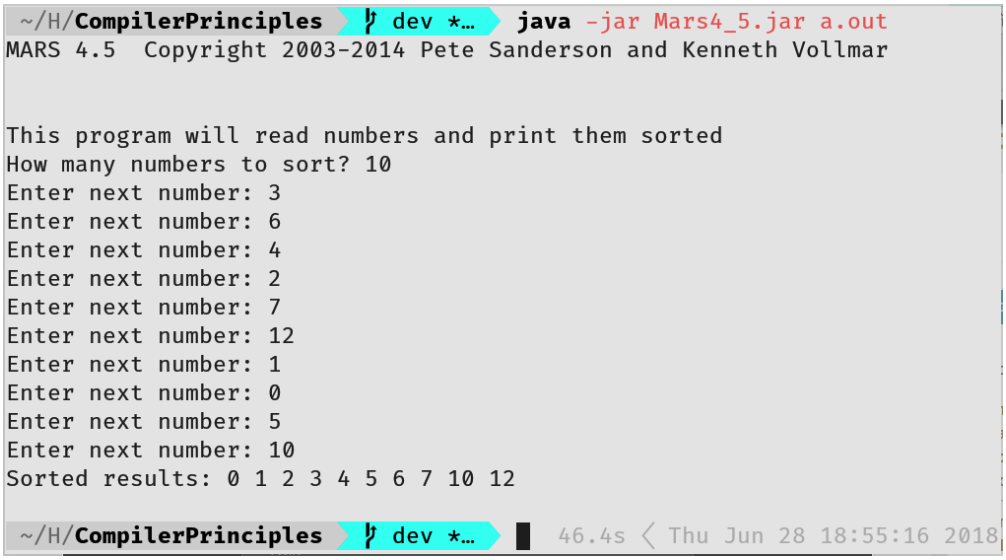

图 5.2: Mars 模拟编译后的程序运行结果

第 6 章　实验心得

在本次实验中，我借助 flex，bison 等工具从零开始制造了一个 Decaf 编译器，在这个过程中我收获了许多。首先，是对于编译器工作原理的理解，从前端到后端，从大体的流程框架到小的细节，我更加深入的理解了编译器是如何工作的，各个数据结构之间是如何转换与协同运作的。通过 flex 与 bison 的编写，我加深了对于课堂上所讲的源程序如何转换词汇表的理解

除了编译原理有关的知识外，我还在编写的过程中对于 C++ 的使用有了更深入一步的了解。经过反复的思考与设计，如何利用一个语言的特性来高效的设计编译器等。最重要的是，在这一个过程中，我体会到了如何完整的设计一个系统，使之可以无缝接合，正常工作，同又不至于耦合度过高，并具有一定的可扩展性。

这次实验并不是一个十分完美的实验，还有许多可扩展的空间，如代码优化，寄存器优化，以及生成 LLVM 可识别的 TAC 等等。在时间有限的情况下，选择需要需要实现的重点内容对我来说也是一次十分宝贵的经验。这些所有的经验在将来的学习过程中也会由很大的帮助。

附录 A　程序使用方法

要编译本程序，需要在 Linux 环境下进行，并保证环境中安装有表1.1 所示的同版本或更高版本的软件。在 Makefile 所在目录下运行 make 就可以自动编译了。

生成的二进制文件为 decafcc，dcc 为使用 bash 脚本对其的包装。在使用时，推荐使用 dcc 这一包装。使用命令为`./dcc <file>`，file 为用于测试的 decaf 文件的路径。

使用上述命令会生成`tokens.out, ast.out, scope.out, tac.out, a.out`这 5 个文件，分别为词法分析结果，语法分析结果，符号表以及作用域分析结果，三地址码中间代码以及目标代码。使用 cat 命令可以直接在终端显示这些文件的内容。由于部分文件使用了 linux 颜色转义字符[1]，因此使用文本编辑器打开极有可能不能正确显示，推荐在现代终端下使用 cat 命令进行输出。使用 Mars 4.5 及以上的版本打开 a.out 文件可以进行目标代码的模拟运行。

[1] `https://misc.flogisoft.com/bash/tip_colors_and_formatting`

附录 B　测试用 Decaf 代码

在本报告中使用的测试用 decaf 文件为一排序程序，程序如下。使用者也可以编写符合 Decaf 语法规范的程序进行测试。

```
int[] ReadArray() {
    int i;
    int num;
    int [] arr;
    int numScores;

    Print("How many numbers to sort? ");
    numScores = ReadInteger();
    arr = NewArray(numScores, int);
    i = 0;
    while (i < arr.length()) {
        Print("Enter next number: ");
        num = ReadInteger();
        arr[i] = num;
        i = i + 1;
    }
    return arr;
}

void Sort(int []arr) {
    int i;
    int j;
    int val;

    i = 1;
    while (i < arr.length()) {
        j = i - 1;
        val = arr[i];
        while (j >= 0) {
            if (val >= arr[j])
                break;
            arr[j + 1] = arr[j];
            j = j - 1;
```

```
34          }
35          arr[j + 1] = val;
36          i = i + 1;
37      }
38  }
39
40  void PrintArray(int []arr) {
41      int i;
42      i = 0;
43      Print("Sorted results: ");
44      while (i < arr.length()) {
45          Print(arr[i], " ");
46          i = i + 1;
47      }
48      Print("\n");
49  }
50
51
52  void main() {
53      int[] arr;
54
55      Print("\nThis program will read numbers and print them sorted\n");
56      arr = ReadArray();
57      Sort(arr);
58      PrintArray(arr);
59  }
```

附录 C　编译器源代码

各个文件的内容及作用如表C.1所示。

表 C.1: caption

文件	作用	文件	作用
Doxyfile	文档生成配置文件	ast_stmt.cpp	语法树语句结点实现
Makefile	Make 文件	ast_stmt.h	语法树语句结点声明
ast.cpp	语法树基类实现	ast_type.cpp	语法树类型结点实现
ast.h	语法树基类声明	ast_type.h	语法树类型结点声明

文件	作用	文件	作用
ast_decl.cpp	语法树声明节点实现	codegen.cpp	TAC 代码生成器实现
ast_decl.h	语法树声明节点声明	codegen.h	TAC 代码生成器声明
ast_expr.cpp	语法树表达式节点实现	dcc	可执行文件 decafcc 包装
ast_expr.h	语法树表达式节点声明	defs.asm	decaf 链接库
errors.cpp	错误处理实现	mips.cpp	Mips 指令生成器实现
errors.h	错误处理声明	mips.h	Mips 指令生成器声明
flow.cpp	CFG 实现	parser.h	语法分析器导出符号
flow.h	CFG 声明	parser.y	语法分析器 bison 文件
hash.h	哈希表声明及实现	printer.cpp	输出管理器实现
list.h	链表声明及实现	printer.h	输出管理器声明
main.cpp	程序入口	scanner.h	词法分析器导出符号
scope.cpp	作用域声明	scanner.l	词法分析器 flex 文件
scope.h	作用域实现	utility.h	工具类声明
tac.cpp	三地址码实现	utility.cpp	工具类实现
tac.h	三地址码声明	tags	索引标签

部分关键源码如下（全部源码过于冗长，此处不再全部放出，详见源文件）：

scanner.l

```
1  /**
2   * @file scanner.l
3   * @brief The lex file to generate scanner
4   *
5   * @details
6   * Scan tokens from stdin.
7   *
8   * @date 2018-06-11
9   * @author Sixu Hu
10  **/
11
12 %{
13 #include <stdio.h>
14 #include <string.h>
15 #include "scanner.h"
16 #include "utility.h"
17 #include "errors.h"
18 #include "parser.h"
19 #include "list.h"
20 #include "printer.h"
21
```

```
22 #define TAB_SIZE 4
23
24 /// @brief current line number and column number
25 static int curLineNum, curColNum;
26
27 /// @brief save all lines for error reporting
28 List<const char*> savedLines;
29
30 /// @brief function called before every actions
31 static void commonAction();
32
33 // install the handle
34 #define YY_USER_ACTION commonAction();
35
36 /// @brief output the scanned token and its corresponding type
37 /// @param type the type of the token, translated to string directly
38 #define OUTPUT(type) LexPrinter::getDefaultPrinter().printToken(yytext, #type);
39
40 %}
41
42  /* Starting states (stackable) */
43 %s NORMAL
44 %x COPY COMMENTS
45 %option stack
46
47  /* Definitions */
48 integerHex          (0[Xx][0-9a-fA-F]+)
49 integer             ([0-9]+)
50 exponent            ([Ee][-+]?{integer})
51 double              ({integer}"."[0-9]*{exponent}?)
52 stringBegin         (\"[^"\n]*)
53 string              ({stringBegin}\")
54 identifier          ([a-zA-Z][a-zA-Z_0-9]*)
55 operator            ([-+/*%=.,;!<>()[\]{}])
56 commentBegin        ("/*")
57 commentEnd          ("*/")
58 commentSingleLine   ("//"[^\n]*)
59
60  /* body ---------------------------------------------------------------- */
61
62 %%
63
64  /* Keywords */
65 "void"          { OUTPUT(tokenVoid       ) return tokenVoid;        }
66 "int"           { OUTPUT(tokenInt        ) return tokenInt;         }
67 "double"        { OUTPUT(tokenDouble     ) return tokenDouble;      }
```

```
68  "bool"          { OUTPUT(tokenBool       ) return tokenBool;        }
69  "string"        { OUTPUT(tokenString     ) return tokenString;      }
70  "null"          { OUTPUT(tokenNull       ) return tokenNull;        }
71  "class"         { OUTPUT(tokenClass      ) return tokenClass;       }
72  "extends"       { OUTPUT(tokenExtends    ) return tokenExtends;     }
73  "this"          { OUTPUT(tokenThis       ) return tokenThis;        }
74  "interface"     { OUTPUT(tokenInterface  ) return tokenInterface;   }
75  "implements"    { OUTPUT(tokenImplements ) return tokenImplements;  }
76  "while"         { OUTPUT(tokenWhile      ) return tokenWhile;       }
77  "for"           { OUTPUT(tokenFor        ) return tokenFor;         }
78  "if"            { OUTPUT(tokenIf         ) return tokenIf;          }
79  "else"          { OUTPUT(tokenElse       ) return tokenElse;        }
80  "return"        { OUTPUT(tokenReturn     ) return tokenReturn;      }
81  "break"         { OUTPUT(tokenBreak      ) return tokenBreak;       }
82  "New"           { OUTPUT(tokenNew        ) return tokenNew;         }
83  "NewArray"      { OUTPUT(tokenNewArray   ) return tokenNewArray;    }
84  "Print"         { OUTPUT(tokenPrint      ) return tokenPrint;       }
85  "ReadInteger"   { OUTPUT(tokenReadInteger) return tokenReadInteger; }
86  "ReadLine"      { OUTPUT(tokenReadLine   ) return tokenReadLine;    }
87
88   /* Operators */
89  "<="            { OUTPUT(tokenLessEqual   ) return tokenLessEqual;   }
90  ">="            { OUTPUT(tokenGreaterEqual) return tokenGreaterEqual;}
91  "=="            { OUTPUT(tokenEqual       ) return tokenEqual;       }
92  "!="            { OUTPUT(tokenNotEqual    ) return tokenNotEqual;    }
93  "&&"            { OUTPUT(tokenAnd         ) return tokenAnd;         }
94  "||"            { OUTPUT(tokenOr          ) return tokenOr;          }
95  {operator}      { OUTPUT(tokenOperator    ) return yytext[0];        }
96  "[]"            { OUTPUT(tokenDims        ) return tokenDims;        }
97
98   /* Constants */
99  "true"|"false"  {
100     OUTPUT(tokenBoolConstant)
101     yylval.boolConstant = (yytext[0] == 't');
102     return tokenBoolConstant;
103 }
104 {integer}       {
105     OUTPUT(tokenIntConstant)
106     yylval.integerConstant = strtol(yytext, NULL, 10);
107     return tokenIntConstant;
108 }
109 {integerHex}    {
110     OUTPUT(tokenIntConstant)
111     yylval.integerConstant = strtol(yytext, NULL, 16);
112     return tokenIntConstant;
113 }
```

```
114 {double}        {
115     OUTPUT(tokenDoubleConstant)
116     yylval.doubleConstant = atof(yytext);
117     return tokenDoubleConstant;
118 }
119 {string}        {
120     OUTPUT(tokenStringConstant)
121     yylval.stringConstant = strdup(yytext);
122     return tokenStringConstant;
123 }
124 {stringBegin}   { ReportError::UntermString(&yylloc, yytext); }
125
126  /* Identifiers */
127 {identifier}    {
128     OUTPUT(tokenIdentifier)
129     if (strlen(yytext) > MAX_IDENTIFIER_LENGTH)
130         ReportError::LongIdentifier(&yylloc, yytext);
131     strncpy(yylval.identifier, yytext, MAX_IDENTIFIER_LENGTH);
132     yylval.identifier[MAX_IDENTIFIER_LENGTH] = '\0';
133     return tokenIdentifier;
134 }
135
136  /* comments */
137 {commentBegin}          { BEGIN(COMMENTS); }
138 <COMMENTS>{commentEnd}  { BEGIN(NORMAL);   }
139 <COMMENTS><<EOF>>       {
140     ReportError::UntermComment();
141     return 0;
142 }
143 <COMMENTS>.         { /* ignore everything in comments */ }
144 {commentSingleLine} { /* ignore sigleline comments */     }
145
146  /* copy line into savedLines */
147 <COPY>.*        {
148     char curLine[512];
149     //strncpy(curLine, yytext, sizeof(curLine));
150     savedLines.Append(strdup(yytext));
151     curColNum = 1;
152     yy_pop_state();
153     // push back all tokens
154     yyless(0);
155 }
156
157  /* new line or tab */
158 <*>\n           {
159     curLineNum++; curColNum = 1;
```

```
160     if (YYSTATE == COPY)
161         savedLines.Append("");
162     else
163         yy_push_state(COPY);
164 }
165 <*>[\t]         { curColNum += TAB_SIZE - curColNum%TAB_SIZE + 1; }
166
167  /* ignore spaces */
168 [ ]+            { }
169
170  /* end of the document */
171 <COPY><<EOF>>   { yy_pop_state(); }
172
173  /* Default (error) */
174 . { ReportError::UnrecogChar(&yylloc, yytext[0]); }
175
176 %%
177
178
179 /* functions ---------------------------------------------------------------- */
180
181 void initScanner() {
182     BEGIN(NORMAL);
183     curLineNum = 1;
184     curColNum = 1;
185     yy_flex_debug = false;
186     yy_push_state(COPY); // copy first line at start
187 }
188
189 static void commonAction() {
190    yylloc.first_line = curLineNum;
191    yylloc.first_column = curColNum;
192    yylloc.last_column = curColNum + yyleng - 1;
193    // yyleng is the length of the token
194    curColNum += yyleng;
195 }
196
197 const char *getNthLine(int num) {
198    if (num <= 0 || num > savedLines.NumElements())
199        return NULL;
200    return savedLines.Nth(num-1);
201 }
```

parser.y

```
1  /**
2   * @file parser.y
```

```
 3   * @brief The yacc file to generate the parser
 4   *
 5   * @details
 6   * build AST from tokens
 7   *
 8   * @date 2018-06-11
 9   * @author Sixu Hu
10  **/
11
12 %{
13 #include "scanner.h"
14 #include "parser.h"
15 #include "errors.h"
16 %}
17
18  /* the token type union*/
19 %union {
20     int              integerConstant;
21     bool             boolConstant;
22     char             *stringConstant;
23     double           doubleConstant;
24     char             identifier[MAX_IDENTIFIER_LENGTH + 1];
25     Decl             *declaration;
26     List<Decl*>      *declarationList;
27     Type             *type;
28     NamedType        *namedType;
29     List<NamedType*> *namedTypeList;
30     FnDecl           *functoinDeclaration;
31     VarDecl          *variableDeclaration;
32     List<VarDecl*>   *variableList;
33     Expr             *expression;
34     List<Expr*>      *expressionList;
35     Stmt             *statement;
36     List<Stmt*>      *statementList;
37     LValue           *lvalue;
38 }
39
40
41 /* terminal token types without value */
42 %token tokenVoid tokenBool tokenInt tokenDouble tokenString tokenClass
43 %token tokenLessEqual tokenGreaterEqual tokenEqual tokenNotEqual tokenDims
44 %token tokenAnd tokenOr tokenNull tokenExtends tokenThis tokenInterface
       tokenImplements
45 %token tokenWhile tokenFor tokenIf tokenElse tokenReturn tokenBreak
46 %token tokenNew tokenNewArray tokenPrint tokenReadInteger tokenReadLine
47
```

```
48 /* terminal token types with value */
49 %token <identifier>      tokenIdentifier
50 %token <stringConstant>  tokenStringConstant
51 %token <integerConstant> tokenIntConstant
52 %token <doubleConstant>  tokenDoubleConstant
53 %token <boolConstant>    tokenBoolConstant
54
55 /* non-terminal token types */
56 %type <expression>          Constant Expr Call OptExpr
57 %type <lvalue>              LValue
58 %type <type>                Type
59 %type <namedType>           OptExt
60 %type <namedTypeList>       OptImpl ImpList
61 %type <declaration>         ClassDecl Decl Field IntfDecl
62 %type <functoinDeclaration> FnDecl FnHeader
63 %type <declarationList>     FieldList DeclList IntfList
64 %type <variableDeclaration> Variable VarDecl
65 %type <variableList>        Formals FormalList VarDecls
66 %type <expressionList>      Actuals ExprList
67 %type <statement>           Stmt StmtBlock OptElse
68 %type <statementList>       StmtList
69
70 /* precedence and associativity */
71 %left     '='
72 %left      tokenOr
73 %left      tokenAnd
74 %nonassoc  tokenEqual tokenNotEqual
75 %nonassoc  '<' '>' tokenLessEqual tokenGreaterEqual
76 %left      '+' '-'
77 %left      '*' '/' '%'
78 %nonassoc  tokenUnaryMinus '!'
79 %nonassoc  '.' '['
80 %nonassoc  tokenLower_Than_Else
81 %nonassoc  tokenElse
82
83  /* body ---------------------------------------------------------------- */
84
85 %%
86 Program     : DeclList {
87                 Program *program = new Program($1);
88                 if (ReportError::NumErrors() == 0)
89                     program->check();
90                 if (ReportError::NumErrors() == 0)
91                     program->generate();
92             }
93             ;
```

```
94
95  DeclList    : DeclList Decl {
96                  ($$=$1)->Append($2);
97              }
98              | Decl {
99                  ($$ = new List<Decl*>)->Append($1);
100             }
101             ;
102
103 Decl        : ClassDecl
104             | FnDecl {
105                 $$=$1;
106             }
107             | VarDecl {
108                 $$=$1;
109             }
110             | IntfDecl
111             ;
112
113 VarDecl     : Variable ';'
114             ;
115
116 Variable    : Type tokenIdentifier {
117                 $$ = new VarDecl(new Identifier(@2, $2), $1);
118             }
119             ;
120
121 Type        : tokenInt {
122                 $$ = Type::intType;
123             }
124             | tokenBool {
125                 $$ = Type::boolType;
126             }
127             | tokenString {
128                 $$ = Type::stringType;
129             }
130             | tokenDouble {
131                 ReportError::Formatted(&@1, "No code gen for doubles");
132                 $$ = Type::errorType;
133             }
134             | tokenIdentifier {
135                 $$ = new NamedType(new Identifier(@1,$1));
136             }
137             | Type tokenDims {
138                 $$ = new ArrayType(concatLocation(@1, @2), $1);
139             }
```

```
140             ;
141
142 IntfDecl    : tokenInterface tokenIdentifier '{' IntfList '}' {
143                 $$ = new InterfaceDecl(new Identifier(@2, $2), $4);
144             }
145             ;
146
147 IntfList    : IntfList FnHeader ';' {
148                 ($$=$1)->Append($2);
149             }
150             | /* empty */ {
151                 $$ = new List<Decl*>();
152             }
153             ;
154
155 ClassDecl   : tokenClass tokenIdentifier OptExt OptImpl '{' FieldList '}' {
156                 $$ = new ClassDecl(new Identifier(@2, $2), $3, $4, $6);
157             }
158             ;
159
160 OptExt      : tokenExtends tokenIdentifier {
161                 $$ = new NamedType(new Identifier(@2, $2));
162             }
163             | /* empty */ {
164                 $$ = NULL;
165             }
166             ;
167
168 OptImpl     : tokenImplements ImpList {
169                 $$ = $2;
170             }
171             | /* empty */ {
172                 $$ = new List<NamedType*>;
173             }
174             ;
175
176 ImpList     : ImpList ',' tokenIdentifier {
177                 ($$=$1)->Append(new NamedType(new Identifier(@3, $3)));
178             }
179             | tokenIdentifier      {
180                 ($$=new List<NamedType*>)->Append(new NamedType(new Identifier(
                        @1, $1)));
181             }
182             ;
183
184 FieldList   : FieldList Field      {
```

```
185                 ($$=$1)->Append($2);
186             }
187             | /* empty */          {
188                 $$ = new List<Decl*>();
189             }
190             ;
191
192 Field       : VarDecl {
193                 $$ = $1;
194             }
195             | FnDecl {
196                 $$ = $1;
197             }
198             ;
199
200
201 FnHeader    : Type tokenIdentifier '(' Formals ')' {
202                 $$ = new FnDecl(new Identifier(@2, $2), $1, $4);
203             }
204             | tokenVoid tokenIdentifier '(' Formals ')' {
205                 $$ = new FnDecl(new Identifier(@2, $2), Type::voidType, $4);
206             }
207             ;
208
209 Formals     : FormalList {
210                             $$ = $1;
211             }
212             | /* empty */ {
213                 $$ = new List<VarDecl*>;
214             }
215             ;
216
217 FormalList  : FormalList ',' Variable {
218                 ($$=$1)->Append($3);
219             }
220             | Variable {
221                 ($$ = new List<VarDecl*>)->Append($1);
222             }
223             ;
224
225 FnDecl      : FnHeader StmtBlock {
226                             ($$=$1)->SetFunctionBody($2);
227             }
228             ;
229
230 StmtBlock   : '{' VarDecls StmtList '}' {
```

```
231                 $$ = new StmtBlock($2, $3);
232             }
233             ;
234
235 VarDecls    : VarDecls VarDecl {
236                 ($$=$1)->Append($2);
237             }
238             | /* empty */ {
239                 $$ = new List<VarDecl*>;
240             }
241             ;
242
243 StmtList    : Stmt StmtList {
244                 $$ = $2; $$->InsertAt($1, 0);
245             }
246             | /* empty */ {
247                 $$ = new List<Stmt*>;
248             }
249             ;
250
251 Stmt        : OptExpr ';' {
252                 $$ = $1;
253             }
254             | StmtBlock
255             | tokenIf '(' Expr ')' Stmt OptElse {
256                 $$ = new IfStmt($3, $5, $6);
257             }
258             | tokenWhile '(' Expr ')' Stmt {
259                 $$ = new WhileStmt($3, $5);
260             }
261             | tokenFor '(' OptExpr ';' Expr ';' OptExpr ')' Stmt {
262                 $$ = new ForStmt($3, $5, $7, $9);
263             }
264             | tokenReturn Expr ';' {
265                 $$ = new ReturnStmt(@2, $2);
266             }
267             | tokenReturn ';' {
268                 $$ = new ReturnStmt(@1, new EmptyExpr());
269             }
270             | tokenPrint '(' ExprList ')' ';' {
271                 $$ = new PrintStmt($3);
272             }
273             | tokenBreak ';' {
274                 $$ = new BreakStmt(@1);
275             }
276             ;
```

```
277
278 LValue      : tokenIdentifier {
279                 $$ = new FieldAccess(NULL, new Identifier(@1, $1));
280             }
281             | Expr '.' tokenIdentifier {
282                 $$ = new FieldAccess($1, new Identifier(@3, $3));
283             }
284             | Expr '[' Expr ']' {
285                 $$ = new ArrayAccess(concatLocation(@1, @4), $1, $3);
286             }
287             ;
288
289 Call        : tokenIdentifier '(' Actuals ')' {
290                 $$ = new Call(concatLocation(@1,@4), NULL, new Identifier(@1,$1
                        ), $3);
291             }
292             | Expr '.' tokenIdentifier '(' Actuals ')' {
293                 $$ = new Call(concatLocation(@1,@6), $1, new Identifier(@3,$3),
                         $5);
294             }
295             ;
296
297 OptExpr     : Expr {
298                 $$ = $1;
299             }
300             | /* empty */ {
301                 $$ = new EmptyExpr();
302             }
303             ;
304
305 Expr        : LValue {
306                 $$ = $1;
307             }
308             | Call
309             | Constant
310             | LValue '=' Expr {
311                 $$ = new AssignExpr($1, new Operator(@2,"="), $3);
312             }
313             | Expr '+' Expr {
314                 $$ = new ArithmeticExpr($1, new Operator(@2, "+"), $3);
315             }
316             | Expr '-' Expr {
317                 $$ = new ArithmeticExpr($1, new Operator(@2, "-"), $3);
318             }
319             | Expr '/' Expr {
320                 $$ = new ArithmeticExpr($1, new Operator(@2,"/"), $3);
```

```
321             }
322             | Expr '*' Expr {
323                 $$ = new ArithmeticExpr($1, new Operator(@2,"*"), $3);
324             }
325             | Expr '%' Expr {
326                 $$ = new ArithmeticExpr($1, new Operator(@2,"%"), $3);
327             }
328             | Expr tokenEqual Expr {
329                 $$ = new EqualityExpr($1, new Operator(@2,"=="), $3);
330             }
331             | Expr tokenNotEqual Expr {
332                 $$ = new EqualityExpr($1, new Operator(@2,"!="), $3);
333             }
334             | Expr '<' Expr {
335                 $$ = new RelationalExpr($1, new Operator(@2,"<"), $3);
336             }
337             | Expr '>' Expr {
338                 $$ = new RelationalExpr($1, new Operator(@2,">"), $3);
339             }
340             | Expr tokenLessEqual Expr {
341                 $$ = new RelationalExpr($1, new Operator(@2,"<="), $3);
342             }
343             | Expr tokenGreaterEqual Expr {
344                 $$ = new RelationalExpr($1, new Operator(@2,">="), $3);
345             }
346             | Expr tokenAnd Expr {
347                 $$ = new LogicalExpr($1, new Operator(@2,"&&"), $3);
348             }
349             | Expr tokenOr Expr {
350                 $$ = new LogicalExpr($1, new Operator(@2,"||"), $3);
351             }
352             | '(' Expr ')' {
353                 $$ = $2;
354             }
355             | '-' Expr  %prec tokenUnaryMinus {
356                 $$ = new ArithmeticExpr(new Operator(@1,"-"), $2);
357             }
358             | '!' Expr {
359                 $$ = new LogicalExpr(new Operator(@1,"!"), $2);
360             }
361             | tokenReadInteger '(' ')' {
362                 $$ = new ReadIntegerExpr(concatLocation(@1,@3));
363             }
364             | tokenReadLine '(' ')' {
365                 $$ = new ReadLineExpr(concatLocation(@1,@3));
366             }
```

```
367             | tokenNew '(' tokenIdentifier ')' {
368                 $$ = new NewExpr(concatLocation(@1,@4),new NamedType(new
                        Identifier(@3,$3)));
369             }
370             | tokenNewArray '(' Expr ',' Type ')' {
371                 $$ = new NewArrayExpr(concatLocation(@1,@6),$3, $5);
372             }
373             | tokenThis {
374                 $$ = new This(@1);
375             }
376             ;
377
378 Constant    : tokenIntConstant {
379                             $$ = new IntConstant(@1,$1);
380             }
381             | tokenBoolConstant {
382                 $$ = new BoolConstant(@1,$1);
383             }
384             | tokenDoubleConstant {
385                 ReportError::Formatted(&@1, "No code gen for doubles");
386                 $$ = new IntConstant(@1,$1);
387             }
388             | tokenStringConstant {
389                 $$ = new StringConstant(@1,$1);
390             }
391             | tokenNull {
392                 $$ = new NullConstant(@1);
393             }
394             ;
395
396 Actuals     : ExprList {
397                 $$ = $1;
398             }
399             | /* empty */ {
400                 $$ = new List<Expr*>;
401             }
402             ;
403
404 ExprList    : ExprList ',' Expr {
405                 ($$=$1)->Append($3);
406             }
407             | Expr {
408                 ($$ = new List<Expr*>)->Append($1);
409             }
410             ;
411
```

```
412 OptElse     : tokenElse Stmt {
413                 $$ = $2;
414             }
415             | /* empty */ %prec tokenLower_Than_Else {
416                 $$ = NULL;
417             }
418             ;
419
420 %%
421
422 void initParser()
423 {
424    debug("parser", "Initializing parser");
425    yydebug = false;
426 }
```

ast.h

```
1  /**
2   * @file ast.h
3   * @brief Defines the base class of all ast nodes
4   *
5   * @details
6   * Each ast node contains a location and parent, location is used for
7   * debugging and reporting errors, parent pointer pointes to the parent node,
8   * each node has a parent node except for the Program node
9   *
10  * @date 2018-06-11
11  * @author Sixu Hu
12  **/
13
14  #ifndef AST_H_
15  #define AST_H_
16
17
18  #include <iostream>
19  #include <fstream>
20  #include <vector>
21  #include <stdlib.h>
22  #include "list.h"
23  #include "utility.h"
24
25  class Scope;
26  class Decl;
27  class Identifier;
28  class Type;
29  class CodeGenerator;
```

```
30
31 /// @brief base class of all ast nodes
32 class AstNode {
33 protected:
34     /// @brief location of the node
35     yyltype *location;
36     /// @brief parent of the node
37     AstNode *parent;
38     /// @brief scope of the node
39     Scope *nodeScope;
40
41 public:
42     AstNode(yyltype loc);
43     AstNode();
44
45     yyltype *GetLocation()   {
46         return location;
47     }
48     void SetParent(AstNode *p)  {
49         parent = p;
50     }
51     AstNode *GetParent()        {
52         return parent;
53     }
54     virtual void check() {}
55
56     /// @brief lookup strategy when finding declarations (shallow or recursive)
57     typedef enum { kShallow, kDeep } lookup;
58
59     /// @brief find declaration for identifier id
60     /// @param id the id to find declaration with
61     /// @param l lookup strategy
62     /// @return the found declare (null for not found)
63     virtual Decl *FindDecl(Identifier *id, lookup l = kDeep);
64
65     virtual Scope *PrepareScope() {
66         return NULL;
67     }
68     template <class Specific> Specific *FindSpecificParent() {
69         AstNode *p = parent;
70         while (p) {
71             if (Specific *s = dynamic_cast<Specific *>(p))
72                 return s;
73             p = p->parent;
74         }
75         return NULL;
```

```
76      }
77
78      /// @brief generate code for ast, converts ast to codegen object
            recursively
79      /// that is, to make the parsed tree a linear code
80      /// @param cg
81      virtual void generate(CodeGenerator *cg) {}
82  };
83
84
85  /// @brief class for identifier (variable/function/class instances and types,
        etc)
86  class Identifier : public AstNode {
87  protected:
88      /// @brief name of the identifier
89      char *name;
90
91      /// @brief cached declaration of the identifier
92      Decl *cached;
93
94  public:
95      /// @brief constructor
96      /// @param loc location of the identifier
97      /// @param name name of the identifier
98      Identifier(yyltype loc, const char *name);
99
100     Decl *GetDeclRelativeToBase(Type *base = NULL);
101
102     const char *getName() {
103         return name;
104     }
105     friend std::ostream &operator<<(std::ostream &out, Identifier *id) {
106         return out << id->name;
107     }
108 };
109
110
111 // This node class is designed to represent a portion of the tree that
112 // encountered syntax errors during parsing. The partial completed tree
113 // is discarded along with the states being popped, and an instance of
114 // the Error class can stand in as the placeholder in the parse tree
115 // when your parser can continue after an error.
116 class Error : public AstNode {
117 public:
118     Error() : AstNode() {}
119 };
```

```
120
121
122 #endif // AST_H_
```

codegen.h

```
1  /**
2   * @file codegen.h
3   * @brief generate code using tac instructions
4   *
5   * @details
6   * tac instructions are defined in tac.h this file parse the ast tree,
7   * generate tac instructions and put all tac instructions in one linear list.
8   *
9   * @date 2018-06-13
10  * @author Sixu Hu
11 **/
12
13 #ifndef CODEGEN_H_
14 #define CODEGEN_H_
15
16 #include <cstdlib>
17 #include <list>
18 #include "tac.h"
19 class FnDecl;
20
21
22 /// @brief built-in functions
23 typedef enum { Alloc, ReadLine, ReadInteger, StringEqual,
24         PrintInt, PrintString, PrintBool, Halt, NumBuiltIns
25 } BuiltIn;
26
27 /// @brief tac code generator class
28 class CodeGenerator {
29 private:
30     //[husixu] stores tac instructions (represented by Instruction class)
31     /// @brief the generated tac code
32     std::list<Instruction *> code;
33     /// @brief current stack offset and global offset
34     int curStackOffset, curGlobalOffset;
35     /// @brief current function
36     BeginFunc *insideFn;
37
38 public:
39     /// @brief som convenient offset constant
40     static const int OffsetToFirstLocal = -8,
41                      OffsetToFirstParam = 4,
```

```
42                      OffsetToFirstGlobal = 0;
43      static const int VarSize = 4;
44
45      /// @brief location of "this" pointer
46      static Location *ThisPtr;
47
48      /// @brief constructor
49      CodeGenerator();
50
51      /// @brief generate a unique global label, does not generate tac
            instructions
52      /// @return label name
53      char *NewLabel();
54
55
56      // these three functions does not generate tac instructions
57      /// @brief generate for a new uniquely named temp variable.
58      /// @return the location of the variable
59      Location *genTempVar();
60      /// @brief generate for a new uniquely named local variable.
61      /// @param varName name of the variable
62      /// @return the location of the variable
63      Location *genLocalVariable(const char *varName);
64      /// @brief generate for a new uniquely named global variable.
65      /// @param varName name of the variable
66      /// @return the location of the variable
67      Location *genGlobalVariable(const char *varName);
68
69      // these three function create a temp var to load a constant value
70      /// @brief generate a tac instruction and a temp var to load a constant int
            .
71      /// @param value the constant int to load
72      /// @return the location of the generated variable
73      Location *genLoadConstant(int value);
74      /// @brief generate a tac instruction and a temp var to load a constant
            string.
75      /// @param str the constant string to load
76      /// @return the location of the generated variable
77      Location *genLoadConstant(const char *str);
78      /// @brief generate a tac instruction and a temp var to load a constant
            location.
79      /// @param str the constant location to load
80      /// @return the location of the generated variable
81      Location *genLoadLabel(const char *label);
82
83
```

```
84      /// @brief generate assign instruction
85      /// @param dst destination variable location
86      /// @param src source variable location
87      void genAssign(Location *dst, Location *src);
88
89      /// @brief generte store instrution to store value to address
90      /// @param addr the address to store into
91      /// @param var the variable location
92      /// @param offset offset in array (0 for no array)
93      void genStore(Location *addr, Location *var, int offset = 0);
94      /// @brief generate load instrution to load value from address
95      /// @param addr the address to load
96      /// @param offset offset to the address
97      /// @return generated variable location
98      Location *genLoad(Location *addr, int offset = 0);
99
100     /// @brief generate biary operator instructions
101     /// @param opName operator name
102     /// @param op1 first operand
103     /// @param op2 second operand
104     /// @return temporary result location
105     Location *genBinaryOp(const char *opName, Location *op1, Location *op2);
106
107     /// @brief push parameterinto stack
108     /// @param param the parameter's location
109     void genPushParam(Location *param);
110     /// @brief pop parameter from stak
111     /// @param numBytesOfParams bytes to pop
112     void genPopParams(int numBytesOfParams);
113
114     /// @brief generate instruction for LCall (call a compile-time address)
115     /// @param label the label to jump to
116     /// @param fnHasReturnValue is function has a return value
117     /// @return the retun value's temp varable's location
118     Location *genLCall(const char *label, bool fnHasReturnValue);
119     /// @brief generate intruction for ACall (call a runtime address)
120     /// @param fnAddr address of the variable pointing to the function address
121     /// @param fnHasReturnValue is function has a return value
122     /// @return the retun value's temp varable's location
123     Location *genACall(Location *fnAddr, bool fnHasReturnValue);
124     /// @brief generate instructions for built-in functions
125     /// @param b the builtin function identifier
126     /// @param arg1 argument 1
127     /// @param arg2 argument 2
128     /// @return the retun value's temp varable's location
129     Location *genBuiltInCall(BuiltIn b, Location *arg1 = NULL, Location *arg2 =
```

```
            NULL);
130
131     /// @brief generate if instruction control flow
132     /// @param test the condition variable location
133     /// @param label the label to jump to
134     void genIfZ(Location *test, const char *label);
135     /// @brief generate goto instruction control flow
136     /// @param label the label to jump to
137     void genGoto(const char *label);
138     /// @brief generate return instruction
139     /// @param val the variable's location whose value is to be returned
140     void genReturn(Location *val = NULL);
141     /// @brief generate instruction for label
142     /// @param label the label name
143     void genLabel(const char *label);
144
145
146     /// @brief generate instructions which marks the satart of the function
147     /// @param fn target function declaration
148     /// @return begining of the function
149     BeginFunc *genBeginFunc(FnDecl *fn);
150     /// @brief generate instructions which marks the end of the function
151     void genEndFunc();
152
153     /// @brief generate virtual table for method
154     /// @param className target class name
155     /// @param methodLabels list of method labels to generate
156     void genVTable(const char *className, List<const char *> *methodLabels);
157
158     Location *genNewArray(Location *numElements);
159     Location *genArrayLen(Location *array);
160     Location *genNew(const char *vTableLabel, int instanceSize);
161     Location *genDynamicDispatch(Location *obj, int vtableOffset, List<Location
            *> *args, bool hasReturnValue);
162     Location *genSubscript(Location *array, Location *index);
163     Location *genFunctionCall(const char *fnLabel, List<Location *> *args, bool
            hasReturnValue);
164
165     // private helper, not for public user
166     Location *genMethodCall(Location *rcvr, Location *meth, List<Location *> *
          args, bool hasReturnValue);
167     void genHaltWithMessage(const char *msg);
168
169     /// @brief translate all generated tac code in to mips instructions
170     void generateFinalCode();
171 };
```

```
172
173
174 #endif // CODEGEN_H_
```

mips.h

```
1  /**
2   * @file mips.h
3   * @brief mips code generator
4   *
5   * @details
6   * controls the generation of mips code, including managing allocation and use
7   * of registers.
8   *
9   * @date 2018-06-13
10  * @author Sixu Hu
11 **/
12
13 #ifndef MIPS_H
14 #define MIPS_H
15
16
17 #include "tac.h"
18 #include "list.h"
19 #include "flow.h"
20 #include <map>
21 class Location;
22
23
24 /// @brief mips instruction generte class
25 class Mips {
26 private:
27     /// @brief all available registers
28     typedef enum {
29         zero, at, v0, v1, a0, a1, a2, a3,
30         s0, s1, s2, s3, s4, s5, s6, s7,
31         t0, t1, t2, t3, t4, t5, t6, t7,
32         t8, t9, k0, k1, gp, sp, fp, ra,
33         f0, f1, f2, f3, f4, f5, f6, f7,
34         f8, f9, f10, f11, f12, f13, f14,
35         f15, f16, f17, f18, f19, f20, f21,
36         f22, f23, f24, f25, f26, f27, f28,
37         f29, f30, f31
38     } Register;
39
40     /// @brief contents of a register
41     struct RegContents {
```

```
42          bool isDirty;
43          Location *var;
44          const char *name;
45          bool isGeneralPurpose;
46          bool mutexLocked;
47          bool canDiscard;
48      } regs[64];
49
50      Register rs, rt, rd;
51
52      /// @brief maps register to location
53      std::map<Register, Location *> registerDescriptor;
54      //std::map<Location*, Register> copyOf_registerDescriptor;
55      int oldestTmpReg;
56
57      typedef enum { ForRead, ForWrite } Reason;
58
59      /// @brief load a word from src into register reg
60      /// @param src the source location
61      /// @param reg the destination variable
62      void fillRegister(Location *src, Register reg);
63      /// @brief store from register to memory location
64      /// @param dst the destination to store
65      /// @param reg the register whose value is to be stored
66      void spillRegister(Location *dst, Register reg);
67      /// @brief remove a discardable value from registerDescriptor
68      void discardValueInRegister(Location *dst, Register reg);
69
70      // register maintenence functions
71      /// @brief get a clean register, clean one if necessary
72      /// @return the cleaned register
73      int regIndexOfNextClean();
74      /// @brief randomlly select a unlocked or dirty register
75      /// @return the selected register
76      int regSelectRandom(); //uses a mutex.
77      /// @brief pick a register for variable
78      /// @param varLoc location of the variable
79      /// @param copyRequired is copying required, if yes, a new register will be
            forced
80      /// @return picked register
81      int regPickRegForVar(Location *varLoc, bool copyRequired);
82      /// @brief clean a register completely (spilling register with unknown
            location)
83      /// @param reg the register to clean
84      void regCleanRegister(Register reg);
85      /// @brief wrapper for clean register
```

```
86      void regCleanForBranch();
87
88      /* register descriptor manipulation and access */
89      /// @brief insert register into descriptor map
90      /// @param varLoc variable location
91      /// @param reg corresponding register
92      void regDescriptorInsert(Location *varLoc, Register reg);
93      /// @brief remove register from descriptor map
94      /// @param varLoc variable location
95      /// @param reg corresponding register
96      void regDescriptorRemove(Location *varLoc, Register reg);
97      /// @brief update variable location for register
98      /// @param varLoc the updated location
99      /// @param reg corresponding register
100     void regDescriptorUpdateRegister(Location *varLoc, Register reg);
101     /// @brief find register according to variable location
102     /// @param varLoc variable location (key)
103     /// @return the found register
104     int regDescriptorLookupRegisterForVar(Location *varLoc);
105     /// @brief find variable according to register
106     /// @param reg the register (key)
107     /// @return found variable
108     Location *regDescriptorGetRegContents(Register reg);
109
110     /// @brief wrapper for the upper function
111     std::map<Register, Location *>::iterator regDescriptorlookupIterForReg(
            Location *varLoc);
112
113     /// @brief generate code for a function call
114     /// @detail used by generate LCall and generateACall, use jal for label and
115     /// jalr for register (dynamic)
116     /// @param dst the destination variable location
117     /// @param function the function to call
118     /// @param isL is static call
119     void generateCallInstr(Location *dst, const char *function, bool isL);
120
121     /// @brief map binary operator to mips instruction
122     static const char *mipsName[BinaryOp::NumOps];
123     /// @brief getter for mipsName
124     /// @param code the binary operator code
125     /// @return mips instruction
126     static const char *tacToMips(BinaryOp::OpCode code);
127
128     Instruction *currentInstruction;
129 public:
130     /// @brief constructor, initializes global constants
```

```
131     Mips();
132
133     /// @brief helper to generate mips instruction
134     /// @param fmt the printf-style format string
135     /// @param ... printf arguments
136     static void generate(const char *fmt, ...);
137     /// @brief discard register for unused value
138     /// @param dst variable location
139     void generateDiscardValue(Location *dst);
140
141     /// @brief assign constant to variable
142     /// @param dst variable location
143     /// @param val the constant to assign
144     void generateLoadConstant(Location *dst, int val);
145     /// @brief assign string constant to vriable
146     /// @param dst variable location
147     /// @param str the string constant to assign
148     void generateLoadStringConstant(Location *dst, const char *str);
149     /// @brief assign label constant to variable
150     /// @param dst variable location
151     /// @param label the label location to assign
152     void generateLoadLabel(Location *dst, const char *label);
153
154     /// @brief load memory content into variable
155     /// @param dst desitnation variable location
156     /// @param reference variable storing address
157     /// @param offset offset to address
158     void generateLoad(Location *dst, Location *reference, int offset);
159     /// @brief store variable content into memory
160     /// @param reference memory address
161     /// @param value variable to store
162     /// @param offset offset to address
163     void generateStore(Location *reference, Location *value, int offset);
164
165     /// @brief copy one variable to an other
166     /// @param dst destination variable location
167     /// @param src source variable location
168     void generateCopy(Location *dst, Location *src);
169
170     /// @brief generate mips code for binary operation
171     /// @param code the binary operator
172     /// @param dst destination variable location
173     /// @param op1 operator one
174     /// @param op2 operator two
175     void generateBinaryOp(BinaryOp::OpCode code, Location *dst,
176             Location *op1, Location *op2);
```

```
177     /// @brief fp version of generateBinaryOp, called by generateBinaryOp
178     /// @see generateBinaryOp()
179     void FP_generateBinaryOp(const char *operation, Location *dst,
180             Location *op1, Location *op2);
181
182     /// @brief generate a label mark
183     /// @param label name of the label
184     void generateLabel(const char *label);
185     /// @brief generate a goto-command
186     /// @param label name of the label to go to
187     void generateGoto(const char *label);
188     /// @brief generate conditional branch using beqz
189     /// @param test condition variable
190     /// @param label label to jump to
191     void generateIfZ(Location *test, const char *label);
192     /// @brief generate return instruction
193     /// @detail save retrun variable to corresponding register and revert
194     /// the stack frame.
195     /// @param returnVal return value
196     void generateReturn(Location *returnVal);
197
198     /// @brief calles's code when entering the function
199     /// @details callee decrease $sp to make space and save current vaules
200     /// of $fp and $ra , and setup new $fp then decrease $sp again to make
201     /// space for all local variables
202     /// @param frameSize size of the stack frame
203     void generateBeginFunction(int frameSize);
204     /// @brief generate ending code of the function, calls generateReturn()
205     void generateEndFunction();
206
207     /// @brief push parameter into stack frame
208     /// @param arg the parameter variable location
209     void generateParam(Location *arg);
210     /// @brief generate call for static function call
211     /// @param result result variable location
212     /// @param label function label name
213     void generateLCall(Location *result, const char *label);
214     /// @brief generate call for dynamic function call
215     /// @param result result variable location
216     /// @param fnAddr variable storing the function address
217     void generateACall(Location *result, Location *fnAddr);
218     /// @brief remove all parameters from stack
219     /// @param bytes bytes to remove
220     void generatePopParams(int bytes);
221
222     /// @brief generate class virtual table
```

```
223     /// @param label class name
224     /// @param methodLabels methods name
225     void generateVTable(const char *label, List<const char *> *methodLabels);
226 
227     /// @brief generate header for the program
228     void generateHeader();
229 
230     /// @brief current instruction
231     class CurrentInstruction;
232 };
233 
234 
235 /// @brief current mips instruction
236 class Mips::CurrentInstruction {
237 public:
238     /// @brief constructor
239     CurrentInstruction(Mips &mips, Instruction *instr)
240         : mips( mips ) {
241         mips.currentInstruction = instr;
242     }
243 
244     /// @brief destructor
245     ~CurrentInstruction() {
246         mips.currentInstruction = NULL;
247     }
248 
249 private:
250     Mips &mips;
251 };
252 
253 
254 #endif // MIPS_H
```

dcc

```
1  #!/bin/bash
2  
3  CURDIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
4  COMPILER="${CURDIR}"/decafcc
5  
6  # check parameters
7  [[ $# != 1 && $# != 3 ]] && {
8      echo "Usage: $(basename "${COMPILER}") <decaf-file> [-o <output-file>] "
9      exit 1
10 }
11 [[ $# == 3 && $2 != "-o" ]] && { echo "$2 not recognized."; exit 1; }
12 [[ $# == 3 && -f $3 ]] && { echo "$3 exists, overwriting ..."; }
```

```
13
14 # define output file, default to a.out
15 OUTPUT=${3:-a.out}
16
17 # check compiler
18 [[ ! -x "$COMPILER" ]] && { echo "$COMPILER is not an executable file"; exit 1;
       }
19
20 # check input file
21 [[ ! -r "$1" ]] && { echo "Cannot find decaf file $1."; exit 1; }
22
23 # compile
24 [[ $(eval "${COMPILER} < $1 > ${OUTPUT}") -ne 0 ]] && {
25     echo "[91mError compiling '$1'.[0m"
26     exit 1;
27 }
28 echo "[92mDone.[0m"
29
30 # link
31 cat "${CURDIR}/defs.asm" >> "${OUTPUT}"
32
33 # $SPIM -trap_file trap.handler -file tmp.asm
34 exit 0;
```

Makefile

```
1  .PHONY: clean strip install
2  CC= g++
3  LD = g++
4  LEX = flex
5  YACC = bison
6
7  COMPILER = decafcc
8  DEPENDES = defs.asm dcc
9  PRODUCTS = $(COMPILER)
10 default: $(PRODUCTS)
11
12 SRCS = ast.cpp ast_decl.cpp ast_expr.cpp ast_stmt.cpp ast_type.cpp scope.cpp \
13     codegen.cpp tac.cpp mips.cpp errors.cpp utility.cpp flow.cpp main.cpp
           printer.cpp
14 OBJS = y.tab.o lex.yy.o $(patsubst %.cpp, %.o, $(filter %.cpp,$(SRCS))) $(
       patsubst %.c, %.o, $(filter %.c, $(SRCS)))
15 JUNK =  *.o lex.yy.c dpp.yy.c y.tab.c y.tab.h *.core core $(COMPILER).purify
       purify.log
16
17 CFLAGS = -g  -Wall -Wno-unused -Wno-sign-compare -Wno-deprecated
18 LEXFLAGS = -d
```

```
19 YACCFLAGS = -dvty
20 LIBS = -lc -lm -lfl
21
22 .yy.o: $*.yy.c
23     $(CC) $(CFLAGS) -c -o $@ $*.cpp
24
25 lex.yy.c: scanner.l  parser.y y.tab.h
26     $(LEX) $(LEXFLAGS) scanner.l
27
28 y.tab.o: y.tab.c
29     $(CC) $(CFLAGS) -c -o y.tab.o y.tab.c
30
31 y.tab.h y.tab.c: parser.y
32     $(YACC) $(YACCFLAGS) parser.y
33 .cpp.o: $*.cpp
34     $(CC) $(CFLAGS) -c -o $@ $*.cpp
35
36 $(COMPILER) :  $(OBJS)
37     $(LD) -o $@ $(OBJS) $(LIBS)
38
39 strip : $(PRODUCTS)
40     strip $(PRODUCTS)
41     rm -rf $(JUNK)
42
43 clean:
44     rm -f $(JUNK) y.output $(PRODUCTS)
45
46 install:
47     mkdir -p build
48     cp $(DEPENDES) $(COMPILER) build/
```

参考文献

[1] 张素琴, 吕映芝, 蒋维杜, and 戴桂兰. 编译原理. 清华大学出版社, 2005.

[2] 曹计昌. *C* 语言与程序设计. 电子工业出版社, 2013.

[3] 王元珍, 曹忠升, and 韩宗芬. *80X86* 汇编语言程序设计. 华中科技大学出版社, 2005.

[4] 王雷. 编译原理课程设计. 机械工业出版社, 2005.

[5] 胡伦俊, 徐兰芳, and 骆婷. 编译原理. 电子工业出版社, 2007.