

# 编译实习MiniC报告

---

罗昊 1700010686

hcc(Hao miniC Compiler) is a simple compiler for MiniC.

It contains 3 parts: eeyore, tigger and risc-v.

Get source code from <https://github.com/vangohao/hcc>

## supported additional C rules besides MiniC

0. 支持空语句(;)
1. 支持逻辑表达式与算术表达式互相自动转换.
2. 支持无返回值调用函数.
3. 支持调用函数时使用表达式作为参数.
4. 支持在程序体内声明函数.
5. 支持C风格多行注释和C++风格单行注释.

## error report

0. 报告语法错误和词法错误及其行号,并给出该处正确的token类型提示.
1. 检查标识符使用,如果使用了未定义的标识符会报错.
2. 检查标识符重复,对于重复定义的报错,函数名与变量名冲突的报错,如果在{}程序块内使用与程序块外同名的变量,则不会报错.
3. 检查函数参数表,对于重复声明但参数表不一致,或定义与声明参数表不一致,或调用时所用的参数表与声明的类型不一致时报错.
4. 检查+,-,\*,/,%运算符对于数组类型变量的不合法操作给出错误提示,这些操作中除了 (int[])+(int), (int)+(int[]), (int[])-(int), 外涉及数组的运算都是不合法的.
5. 检查对数组变量的赋值,无法将数值赋给数组变量.
6. 检查a[b]使用,如果a不是数组变量,会报错.

## eeyore

0. 使用flex,bison和C++构建,将输入的MiniC代码转换为Eeyore三地址代码.
1. 使用STL的map模板制作符号表,使用链表串联内外层程序块的符号表.
2. 使用回填法构建eeyore中的标号及goto语句.

## Tigger

### 代码结构

代码由analyz.cpp/h lexer.l tigger.y tigger.cpp/h main.cpp 构成

analyz.cpp: 包含三个类的定义: 1.Expression类,每个Expression对象对应一条Tigger/RiscV指令; 2.Func类,每个Func对象对应一个函数; 3.Analyz类,单例类,负责整体处理工作以及函数间优化工作.

analyz.h: 包含上述三个类的声明.

**lexer.l:** 词法分析器,负责词法分析,其中对于变量,区分全局变量,局部变量和形参,使用Analyz中的vcount属性为变量分配id,并包含在yyval中.

**tigger.y:** 语法分析器,负责语法分析,负责将Eeyore语句转换为Expression对象, 以及负责Func类对象创建, Func类对象负责tigger的主要函数体内工作.

**tigger.cpp:** 包含一些公共函数.

**tigger.h:** 公共头文件.

**main.cpp:** 主程序函数和少量全局变量.

## 类结构



### 1. Expression类:

#### 活性分析

以语句为基本块进行自下而上的活性分析

#### 寄存器分配

使用图着色算法进行寄存器分配.

利用活性分析的结果构造冲突图,兼用邻接表和邻接矩阵表示冲突图.

图染色有四种操作:

- 1.简化 对于度数小于k的传送无关的点,将其从图中删除并放入栈中.
- 2.合并 对于一条传送指令,对其关联的两个点在保守规则下合并.
- 3.冻结 冻结一条传送指令,将他当作非传送指令.
- 4.溢出 将图中度数最大的点放入栈中,并从图中删除..

操作按1.2.3.4的优先级进行,直到图中不再有点.

将栈中元素依次弹出,并为其分配颜色,如果无颜色可用,则将该顶点真实溢出.

若有真实溢出发生,则重写程序,为溢出的变量分配栈空间,加入load语句和store语句来读取和写回,并创建临时变量进行运算.

#### 其他说明

Tigger不再创建符号表,因为其中所有变量名称的作用域都是全局,所以直接由词法分析器为其分配一个id(>27). 全局变量和栈数组不分配寄存器,直接存到内存中.

#### 预着色节点的处理

Tigger有28个寄存器,给其确定id为0-27,以作为预着色节点,在活性分析中与其他变量处于同等地位. 在图着色时也可与与其相关的传送节点合并.

## 形参的处理

函数参数及返回值需要预着色,函数体的开头会创建临时变量,使用传送指令将预着色节点传送到临时变量作为形参,以避免形参长期占用a开头寄存器.

## 调用函数

该部分tigger.y代码:

```
Expression:
| ...
| PARAM Symbol
{AnalyzeInstance.currentFunc().CallParam($2,1);}
| PARAM INTEGER
{AnalyzeInstance.currentFunc().CallParam($2,0);}
| Symbol '=' CALL FUNCTION
{AnalyzeInstance.currentFunc().CallFunc($1,$4);}
| ...
```

该部分analyz.cpp代码:

```
void Func::CallParam(int v,int t)
{
    if(t == 1)
    {
        new Expression(MoveRR, {(int)(a0)+paramToCallWithCount}, {v}, {});
    }
    else
    {
        new Expression(MoveRI, {(int)(a0)+paramToCallWithCount}, {}, {v});
    }
    paramToCallWithCount++;
}
void Func::CallFunc(int v,string f)
{
    vector<int> paramvec;
    paramvec.push_back(int(a0));
    for(int i = 1; i<paramToCallWithCount; i++)
    {
        paramvec.push_back((int)(a0) + i);
    }
    new Expression(Call, {(int)(a0)}, paramvec, {}, f);
    new Expression(MoveRR, {v}, {(int)(a0)}, {});
    paramToCallWithCount = 0;
}
```

Func类设置了一个计数器计数当前传入参数的个数. 计数器初始化为0, 每扫描并处理一个Param语句, 将计数器值加一.

对于Param x语句,将其处理为 a0 = x(对应Expression的类型为MoveRR), 其中0可能为0-7,是传参计数器的当前

值.

由Eeyore的构成,在Param语句和对应的Call语句之间不会有其他会改变参数寄存器值的语句,而Call语句的right属性会被设置成{a0,a1,...,at},其中t为参数的个数-1;

因此活性分析的结果会阻止Param语句和Call语句之间的其他无关变量被分配为a0-at寄存器.

合并预着色点相关传送节点的例子

MiniC代码

```
int v0;
v0 = getint();
int v1;
v1 = func(v0);
int v2;
v2 = putint(v1+1);
return v2;
```

tigger代码

```
call getint
call func
a0 = a0 + 1
call putint
return
```

riscv64代码

```
main:
    add    sp,sp,-16
    sd     ra,8(sp)
    call   getint
    call   func
    addiw  a0,a0,1
    call   putint
    ld     ra,8(sp)
    addi   sp,sp,16
    jr     ra
    .size  main, .-main
```

变量大量合并,只用了a0,没有一条move指令.