Step 12 Report

刘轩奇 2018011025

2020年10月9日

1 工作内容

本人选择不使用辅助工具 ANTLR 因此自己实现了 lexer 和 parser。

1.1 文件说明

`montLexer.h/cpp` 词法分析器；

`montParser.h/cpp` 语法分析器；

`montConceiver.h/cpp` 产生中间代码；

`montAssembler.h/cpp` 从中间代码产生汇编代码；

`montLog.h` 记录编译错误信息；

`montCompiler.cpp` MiniDecaf 编译器，包含主函数，编译成功返回 0 否则返回 -1，并将错误信息输出到 `std::cerr`。

`refSyntax.txt` 参考语法。

1.2 本步骤完成的工作

1 词法分析 添加了左右方括号对应的 Token。

2 句法分析 修改了以下的树节点：

* `globdecl` 表示全局变量的声明，添加了对数组类型的支持。
* `declaration` 表示局部变量的声明，添加了对数组的支持。
* `postfix` 同指导书上说明。

3 产生中间代码 添加了以下两条中间代码：

* `SWAP` 交换栈顶上的两个元素。
* `COMMENT` 带有一个字符串参数，表示生成注释，用于调试。

在类型分析的过程中添加了对数组的支持，同时在类型分析中加上了对类型内存空间的分析，基本类型和指针类型空间均为4，而数组类型的空间由数组长度和数组元素类型决定。对数组取下标后的表达式均为左值，且其类型为原数组的元素类型。

另外值得注意的是，在栈内存（局部变量）中，数组的起始位置应当是靠近栈顶的位置，即不是插入栈中的第一个地址，而是最后一个地址。

此外还添加了数组有关的运算符操作类型检查、函数调用和返回值类型检查等等细节，此处不再赘述。

4 产生汇编代码 添加了中间代码 SWAP 对应的汇编代码：`lw t1, 4(sp); lw t2, 0(sp); sw t1, 0(sp); sw t2, 4(sp);` ，以及注释对应的汇编代码（井号表示），其余没有变化。

2 思考题

1 设有以下几个函数，其中局部变量 `a` 的起始地址都是 `0x1000`(4096)，请分别给出每个函数的返回值（用一个常量 minidecaf 表达式表示，例如函数 `A` 的返回值是 `*(int*)(4096 + 23 * 4)`）。

```
int A() {
    int a[100];
    return a[23];
}

int B() {
    int *p = (int*) 4096;
    return p[23];
}

int C() {
    int a[10][10];
    return a[2][3];
}

int D() {
    int *a[10];
    return a[2][3];
}

int E() {
    int **p = (int**) 4096;
    return p[2][3];
}
```

答 B: `*(int*)(4096+23*4)`, C: `*(int*)(4096+23*4)`, D: `*(int*)((int)(*(int**)(4096+2*4))+3*4)`, E: `*(int*)((int)(*(int**)(4096+2*4))+3*4)`.

2 C 语言规范规定，允许局部变量是可变长度的数组（Variable Length Array，VLA），在我们的实验中为了简化，选择不支持它。请你简要回答，如果我们决定支持一维的可变长度的数组 (即允许类似 `int n = 5; int a[n];` 这种，但仍然不允许类似 `int n = ...; int m = ...; int a[n][m];` 这种)，而且要求数组仍然保存在栈上（即不允许用堆上的动态内存申请，如`malloc`等来实现它），应该在现有的实现基础上做出那些改动？

答 可以在预分配的栈帧中使用一个指针和一个整数来记录实际运行栈中的可变长度数组。例如，编译器预先已经知道某局部变量`a`是可变长度数组，则在预分配栈帧中留下两个位置`a_pos`和`a_len`来记录，实际运行时，根据实际数组大小和地址，将`a_pos`置为指向实际地址的指针，`a_len`为数组实际长度，则要调用`a`时可以根据这两个值来间接地获取`a`的某下标元素的值或地址。