2016-2017学年第1学期

**实 验 报 告**



* 课程名称:编程语言原理与编译
* 实验项目:期末大作业
* 专业班级\_\_\_计算1501\_\_
* 学生学号\_\_\_31501337,\_\_31502176\_\_\_
* 学生姓名\_\_\_刘坤，吕奇伦\_\_\_
* 实验指导教师: 郭鸣

简介

本次实现了带类型检查的MicroC语言编译器。

**背景**

**编译原理是计算机专业的一门重要专业课，旨在介绍编译程序构造的一般原理和基本方法。内容包括语言和文法、词法分析、语法分析、语法制导翻译、中间代码生成、存储管理、代码优化和目标代码生成。如今快餐化的开发，本课程少有人研究，但是这不影响本课程在计算机领域的重要性，因此本课程大作业就是使用F#语言来实现一个micro C语言，并且做类型检查。**

**目的与目标**

实现一个较为基础的micro C语言，我们小组设计了以下功能：

1. 编译：将micro c程序编译到栈式虚拟机指令，编译保存并检查语法错误
2. 类型检查：检查基本的if，while，赋值语句，基本的运算语句等
3. 语法：主函数，带有随机空格的空主函数、带有随机空格的空主函数、二元算数表达式、一元算数表达式、返回空、返回表达式、if语句、带参数的函数、逻辑运算、if-else语句、局部声明、函数调用、赋值语句、数组。

**基本语法**

## 分号 ;

在程序中，分号是语句结束符。也就是说，每个语句必须以分号结束。它表明一个逻辑实体的结束。

例如，下面是两个不同的语句：

printf(1);  
return 0;

## 标识符

标识符是用来标识变量、函数，或任何其他用户自定义项目的名称。一个标识符以字母 A-Z 或 a-z 或下划线 \_ 开始，后跟零个或多个字母、下划线和数字（0-9）。

标识符内不允许出现标点字符，比如 @、$ 和 %。C 是**区分大小写**的编程语言。因此，在 C 中，*Manpower* 和 *manpower* 是两个不同的标识符。

空格

词法分析时，空格用来分隔两个词元，因此如下条语句：

int a;

int和a间必须有一个空格，否则编译器会认为是一个标识符。

变量

本编译器支持的变量有

|  |  |
| --- | --- |
| 类型 | 声明方式 |
| 整型 | int a; |
| 字符型 | char a; |
| 指针 | int \* a; |

**运算符**

## 算术运算符

下表显示了支持的所有算术运算符。假设变量 **A** 的值为 10，变量 **B** 的值为 20，则：

|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 实例 |
| + | 把两个操作数相加 | A + B 将得到 30 |
| - | 从第一个操作数中减去第二个操作数 | A - B 将得到 -10 |
| \* | 把两个操作数相乘 | A \* B 将得到 200 |
| / | 分子除以分母 | B / A 将得到 2 |
| % | 取模运算符，整除后的余数 | B % A 将得到 0 |
| ++ | 自增运算符，整数值增加 1 | A++ 将得到 11 |
| -- | 自减运算符，整数值减少 1 | A-- 将得到 9 |

## 关系运算符

下表显示了支持的所有关系运算符。假设变量 **A** 的值为 10，变量 **B** 的值为 20，则：

|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 实例 |
| == | 检查两个操作数的值是否相等，如果相等则条件为真。 | (A == B) 不为真。 |
| != | 检查两个操作数的值是否相等，如果不相等则条件为真。 | (A != B) 为真。 |
| > | 检查左操作数的值是否大于右操作数的值，如果是则条件为真。 | (A > B) 不为真。 |
| < | 检查左操作数的值是否小于右操作数的值，如果是则条件为真。 | (A < B) 为真。 |
| >= | 检查左操作数的值是否大于或等于右操作数的值，如果是则条件为真。 | (A >= B) 不为真。 |
| <= | 检查左操作数的值是否小于或等于右操作数的值，如果是则条件为真。 | (A <= B) 为真。 |

## 指针

int \* a;

上面就是指针的申明，a就是一个地址，而 \*a 用来调用指针指向的那个变量。

## 取地址符号 &

& res；

如果res是一个变量，那上面这个语句就是取res这个变量的地址，取地址符和指针经常成队出现，如下面这个实例：

void main(int n) {

int i;

i = 0;

while (i < n) { /\* Outer n \*/

int n;

fac(i, &n);

print n; /\* Inner n \*/

i = i + 1;

}

print n;

}

void fac(int n, int \*res) {

if (n == 0) /\* fac's n \*/

\*res = 1;

else {

int tmp;

fac(n-1, &tmp);

\*res = tmp \* n;

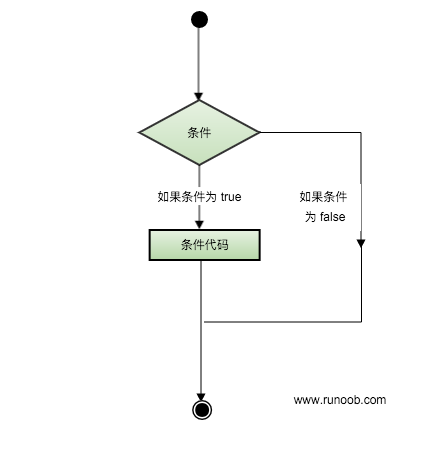
}

}

## 判断

判断结构要求程序员指定一个或多个要评估或测试的条件，以及条件为真时要执行的语句（必需的）和条件为假时要执行的语句（可选的）。

任何**非零**和**非空**的值假定为 **true**，把**零**或 **null** 假定为 **false**。

下面是大多数编程语言中典型的判断结构的一般形式：

## if 语句

if(boolean\_expression)  
{  
 /\* 如果布尔表达式为真将执行的语句 \*/  
}

如果布尔表达式为 **true**，则 if 语句内的代码块将被执行。如果布尔表达式为 **false**，则 if 语句结束后的第一组代码（闭括号后）将被执行。

## if...else 语句

if(boolean\_expression)  
{  
 /\* 如果布尔表达式为真将执行的语句 \*/  
}  
else  
{  
 /\* 如果布尔表达式为假将执行的语句 \*/  
}

如果布尔表达式为 **true**，则执行 **if** 块内的代码。如果布尔表达式为 **false**，则执行 **else** 块内的代码。

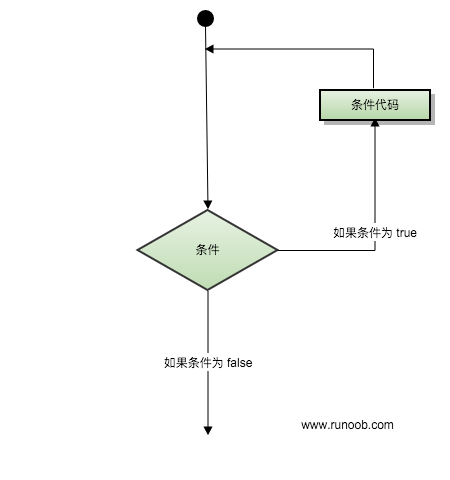
## 嵌套 if 语句

if( boolean\_expression 1)  
{  
 /\* 当布尔表达式 1 为真时执行 \*/  
 if(boolean\_expression 2)  
 {  
 /\* 当布尔表达式 2 为真时执行 \*/  
 }  
}

## 循环

有的时候，我们可能需要多次执行同一块代码。一般情况下，语句是按顺序执行的：函数中的第一个语句先执行，接着是第二个语句，依此类推。

编程语言提供了更为复杂执行路径的多种控制结构。

循环语句允许我们多次执行一个语句或语句组，下面是大多数编程语言中循环语句的流程图：

## while 循环

while(condition)  
{  
 statement(s);  
}

在这里，**statement(s)** 可以是一个单独的语句，也可以是几个语句组成的代码块。**condition** 可以是任意的表达式，当为任意非零值时都为 true。当条件为 true 时执行循环。

当条件为 false 时，程序流将继续执行紧接着循环的下一条语句。

****输出语句****

print n;

n可以是整型也可以是字符串。

## 换行符号

println;

## 函数

函数是一组一起执行一个任务的语句。每个程序都至少有一个函数，即主函数 **main()** ，所有简单的程序都可以定义其他额外的函数。

您可以把代码划分到不同的函数中。如何划分代码到不同的函数中是由您来决定的，但在逻辑上，划分通常是根据每个函数执行一个特定的任务来进行的。

函数**声明**告诉编译器函数的名称、返回类型和参数。函数**定义**提供了函数的实际主体。

## 定义函数

函数定义的一般形式如下：

return\_type function\_name( parameter list )  
{  
 body of the function  
}

函数由一个函数头和一个函数主体组成。下面列出一个函数的所有组成部分：

* **返回类型：**一个函数可以返回一个值。**return\_type** 是函数返回的值的数据类型。有些函数执行所需的操作而不返回值，在这种情况下，return\_type 是关键字 **void**。
* **函数名称：**这是函数的实际名称。函数名和参数列表一起构成了函数签名。
* **参数：**参数就像是占位符。当函数被调用时，您向参数传递一个值，这个值被称为实际参数。参数列表包括函数参数的类型、顺序、数量。参数是可选的，也就是说，函数可能不包含参数。
* **函数主体：**函数主体包含一组定义函数执行任务的语句。

## ****类型检查****

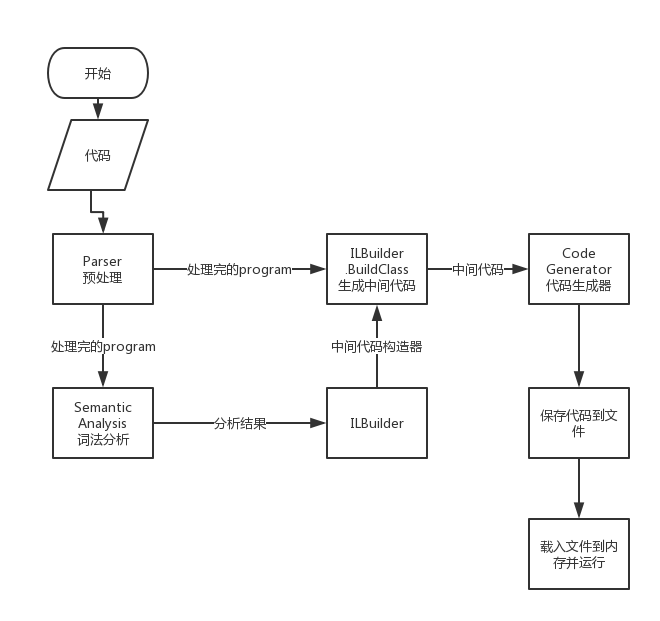
本项目类型检查是编译器之外的模块进行，操作如下：

1. 请书写一个合法的microC文件
2. 使用“fsi -r FsLexYacc.Runtime.dll Absyn.fs CPar.fs CLex.fs Parse.fs Machine.fs Comp.fs typedFun.fs ParseAndComp.fs ”语句来启动类型检查程序
3. “open ParseAndComp;; checkType "要检查的程序名";;”用来检查类型
4. “ #q;;”用来退出类型检查系统

特别注意如果程序不能通过语法分析，那么将不会进行类型检查；

如果类型检查通过不会有任何文字输出，只有当类型检查失败，才会有提示。

**架构设计**



**测试**

**心得体会**

**刘坤：**

**上完编译原理之后其实是比较懵逼的，这是一门很低层的学科，他做了将高级语言向机器语言转换的工作，使得程序员不需要写汇编语言或者更低级的01序列。这门课非常难，以至于长时间处于不能理解的状态。但是通过对每一个实验的学习，我认识了这门课程的冰山一角。在学期末，大概一个月的时间完成了类型检查模块的编写，这个模块当然不是很完美，特别是一开始永远检查不到错误，真让人沮丧。然后，说实话fsharp这门语言真的非常难学，没有c语言逻辑清晰，也没有java那么好理解，希望是真心希望老师以后开着门课不要选择fsharp，入门实在是太难了。还有，这门课最好还是考试吧，由于比较偏理论，其实自己开发一个编译器成功与否都是未知的。**

**计算评价表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 一般 | 中 | 良 | 优 |
| 编译并在内存中运行 |  |  | o |  |
| 编译保存并执行 |  |  |  | o |
| 编译: 检测语法错误 |  |  | o |  |
| 二元表达式 |  |  | o |  |
| 二元算数表达式 |  |  |  | o |
| 语法: if-else语句 |  |  |  | O |
| 语法：while语句 |  |  |  | o |
| 语法: 数组 |  |  | o |  |
| 类型检查 |  | o |  |  |

**分工表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 姓名 | 班级 | 学号 | 任务 | 比重 |
| 刘坤 | 计算1501 | 31501337 | 大部分编码 | 0.95 |
| 吕奇伦 | 计算1501 |  | 测试及文档 | 0.95 |