实验报告1

- 编译原理实验一
- 徐秋灵 141130112

完成内容

- 语法错误的识别
- 文法错误的识别
- 16进制8进制的识别
- 指数浮点形式
- 两种注释的识别
- 字符串的识别
- 即所有要求内容和附加内容

使用方式

- make testl
- 样例在code/lexical_testcase中,编译器输出文件为lexical_result.txt

报告正文

技术难点

- 主要用了flex的宏模式来识别字符串和注释,即在识别注释时切换到另一种模式进行特别的识别.
- 使用了变长的C宏命令方便简化编写程序
- 使用了静态结构上动态建立语法树来加快程序的速度
- 使用了优先级符号来消除BISON程序二义性

实验报告2

- 编译原理实验二
- 徐秋灵 141130112

完成内容

- 函数表
- 变量表
- SDT框架
- 类型表
- 结构表
- 除Struct结构等价外的所有错误检查

使用方式

- make tests
- 样例在code/syntax_testcase中,编译器输出文件为lexical_result.txt

报告正文

实验步骤

首先，通过建立语法树遍历-动作响应框架，使得动作能很方便的加入语法节点的遍历，通过插入动作节点和定义遍历方式属性使得可以方便的传递继承属性与综合属性。

然后，通过建立变量环境栈，把所有变量的声明和定义，（包括函数和结构中的）综合到同一张表中，通过对变量环境栈的维护的搜索，查找最近邻变量和发现变量冲突。

通过建立类型表，对所有类型（数组，基础类型，结构）实现统一的查重与引用。

在建立的结构上，通过对属性的定义实现错误处理的发现与错误恢复。

技术难点

如何把SDT框架写得简洁好用，通过大量的宏定义，宏展开，考虑到大部分属性是S属性和L属性，所以通过合理定义前序遍历动作，中序遍历动作和后序遍历动作实现精简的SDT。

数组的定义需要对节点进行逆序扫描，来与继承属性一致，数组的访问需要顺序扫描节点，来与继承属性一致。

合理的文件架构使得写代码和修改变得更为方便，通过 .h 的宏声明使得所有文件只需包含公共头文件即可。

编写环境栈时，给环境设定4中种类，一种是代码块类型，一种是函数申明类型，一种是函数定义类型，一种是结构体定义类型，即新增变量都可以直接在该框架下进行，不用额外增加代码机制来实现。

实验结果

```
File: ./syntax_testcase/testcase01.in
Error type 1 at Line 4: Undefined Variable "j"
Error type 6 at Line 4: The left-hand side of an assignment must be a variable.

File: ./syntax_testcase/testcase02.in
Error type 2 at Line 4: Undefined function "inc"

File: ./syntax_testcase/testcase03.in
Error type 3 at Line 4: Redefined Variable "i"

File: ./syntax_testcase/testcase04.in
Error type 4 at Line 5: Redefined function  "func"

File: ./syntax_testcase/testcase05.in
Error type 5 at Line 4: Type mismatched for assignment.

File: ./syntax_testcase/testcase06.in
Error type 6 at Line 4: The left-hand side of an assignment must be a variable.

File: ./syntax_testcase/testcase07.in
Error type 7 at Line 4: Type mismatched for operands

File: ./syntax_testcase/testcase08.in
Error type 8 at Line 4: Type mismatched for return

File: ./syntax_testcase/testcase09.in
Error type 9 at Line 8: Function "func" is not applicable for arguments
```

```
File: ./syntax_testcase/testcase101opt.in
Error type 18 at Line 1: Undefined function "func"

File: ./syntax_testcase/testcase102opt.in
Error type 19 at Line 6: Inconsistent declaration of function "func"
Error type 18 at Line 5: Undefined function "func"

File: ./syntax_testcase/testcase103opt.in

File: ./syntax_testcase/testcase104opt.in
Error type 3 at Line 9: Redefined Variable "i"

File: ./syntax_testcase/testcase105opt.in
Error type 5 at Line 15: Type mismatched for assignment.

File: ./syntax_testcase/testcase106opt.in
Error type 5 at Line 14: Type mismatched for assignment.

File: ./syntax_testcase/testcase10.in
Error type 10 at Line 4: it is not an array

File: ./syntax_testcase/testcase11.in
Error type 11 at Line 4: "i" is not a function

File: ./syntax_testcase/testcase12.in
Error type 12 at Line 4: array index is not an integer

File: ./syntax_testcase/testcase13.in
Error type 13 at Line 8: Illegal use of "."

File: ./syntax_testcase/testcase14.in
Error type 14 at Line 8: Non-existent field "n"
Error type 7 at Line 8: Type mismatched for operands

File: ./syntax_testcase/testcase15.in
Error type 16 at Line 4: Redefined field "x"

File: ./syntax_testcase/testcase16.in
Error type 16 at Line 5: Duplicated name "Position"

File: ./syntax_testcase/testcase17.in
Error type 17 at Line 3: Undefined structure "Position"
```

实验报告3

- 编译原理实验三
- 徐秋灵 141130112

完成内容

- 中间代码翻译框架

- 条件控制语句的翻译
- 函数调用返回的翻译
- 多维数组的翻译
- 所有实验要求

使用方式

- make testi
- 或者 make && ./parser [需要翻译的文件] > [输出文件]
- 样例在code/intercode_testcase中,编译器输出文件为intercode_result.txt

报告正文

实验步骤

在实验二的基础上，在动作响应框架内加入翻译动作。

使用双向链表实现中间代码的插入合并，在链表上构建了中间代码翻译框架：

1. 将中间代码符号抽象为以下几类：变量，代码块，代码块导出变量。从而在翻译时根据不同类别专门翻译。
2. 在单个语法节点上加入多个动作槽，使得一个节点可以同时响应语义分析动作和中间代码生成动作。
3. 以变量和常量为基本单位，使用统一调用构建基本句子；以句子和代码块为单位，使用统一调用构建复合代码块。
4. 维护符号表、函数表，环境表，类型表，使其具有中间代码生成所需功能。

技术难点

1. 部分语义比如数组，作为左值和右值时有不同的含义，对于这种情况，需要生成两套代码分别对应不同的情况。
2. 变量与代码块的关系要十分清楚。比如中间变量，有些代码本是为了产生这些变量，是要附着在变量上的，即使用变量前，要先运行附着的代码，但是，对于用户指定的变量来说，每一次引用的附着代码可能并不相同，因此不能直接把代码依附到符号表中的变量上，而是要为每一次引用创建一个新的中间代码变量，来加入依附的代码。

实验结果

File: ./intercode_testcase/testcase01.in

```
FUNCTION main :
READ t1
t0 := t1
t2 := #0
IF t0 <= #0 GOTO l0
t2 := #1
LABEL l0 :
IF t2 != #1 GOTO l4
WRITE #1
GOTO l5
LABEL l4 :
t3 := #0
```

```
IF t0 >= #0 GOTO l1
t3 := #1
LABEL l1 :
IF t3 != #1 GOTO l2
WRITE #-1
GOTO l3
LABEL l2 :
WRITE #0
LABEL l3 :
LABEL l5 :
RETURN #0

File: ./intercode_testcase/testcase01_opt.in
Error type -1 at Line 6: Code contains variables or parameters of structure type

File: ./intercode_testcase/testcase02.in

FUNCTION fact :
PARAM t0
t1 := #0
IF t0 != #1 GOTO l0
t1 := #1
LABEL l0 :
IF t1 != #1 GOTO l1
RETURN t0
GOTO l2
LABEL l1 :
t2 := t0 - #1
ARG t2
t3 := CALL fact
t4 := t0 * t3
RETURN t4
LABEL l2 :
FUNCTION main :
READ t7
t5 := t7
t8 := #0
IF t5 <= #1 GOTO l3
t8 := #1
LABEL l3 :
IF t8 != #1 GOTO l4
ARG t5
t9 := CALL fact
t6 := t9
GOTO l5
LABEL l4 :
t6 := #1
LABEL l5 :
WRITE t6
RETURN #0

File: ./intercode_testcase/testcase02_opt.in

FUNCTION add :
```

```
PARAM t0
t1 := #4 * #0
t1 := t0 + t1
t1 := *t1
t2 := #4 * #1
t2 := t0 + t2
t2 := *t2
t3 := t1 + t2
RETURN t3
FUNCTION main :
DEC t4 8
DEC t5 8
t6 := #0
t7 := #0
LABEL l4 :
t8 := #0
IF t6 >= #2 GOTO l0
t8 := #1
LABEL l0 :
IF t8 != #1 GOTO l5
LABEL l2 :
t9 := #0
IF t7 >= #2 GOTO l1
t9 := #1
LABEL l1 :
IF t9 != #1 GOTO l3
t10 := #4 * t7
t10 := t4 + t10
t11 := t6 + t7
*t10 := t11
t12 := t7 + #1
t7 := t12
GOTO l2
LABEL l3 :
t13 := #8 * #0
t13 := t5 + t13
t14 := #4 * t6
t14 := t13 + t14
ARG t4
t15 := CALL add
*t14 := t15
t16 := #8 * #0
t16 := t5 + t16
t17 := #4 * t6
t17 := t16 + t17
t17 := *t17
WRITE t17
t18 := t6 + #1
t6 := t18
t7 := #0
GOTO l4
LABEL l5 :
RETURN #0
```