华中科技大学

编译技术

编译技术实验

编 译 技 术 课 程 组

课程设置目的和要求——课程要求

- 总体学习目标

1. 熟悉编译程序的总体结构
2. 熟悉编译程序各组成部分及其任务
3. 编译过程各阶段所要解决的问题及其采用的方法和技术
4. 掌握关键算法的工作原理

- 能力要求

1. 掌握程序变换基本概念、问题描述和处理方法
2. 增强理论结合实际能力
3. 培养“问题→形式化描述→计算机化”的问题求解过程
4. 使学生在系统级上认识算法和系统的设计，培养系统能力

课程设置目的和要求——实验要求

- 实验形式
  - 分析、设计、编写、调试、测试程序
  - 系统验收
  - 撰写实验报告
- 实验内容
  - 词法语法分析器的设计与实现　　8学时
  - 符号表设计与语义分析　　8学时
  - 中间代码生成与优化　　8学时
  - 目标代码生成　　8学时

    （建议在自己的CPU上执行）

课程设置目的和要求——考试要求

- 实验评分标准

– 词法、语法分析　　20分
– 符号表与语义计算　　20分
– 中间代码生成与优化　　20分（其中优化5分）
– 目标代码生成　　20分
– 实验报告　　20分

实验内容

- **题目：**XX语言编译器设计与实现（请为自己的编译器命名）
- **源语言定义：**或采用教材中Decaf语言，或采用C语言(或C++语言或C#语言或JAVA语言)部分关键语法规则。源语言要求至少包含的语言成分如下：
- 数据类型至少包括char类型、int类型和float类型，字符串作为可选项；
- 基本运算至少包括算术运算、比较运算、自增自减运算和复合赋值运算；
- 控制语句至少包括if语句、while语句和break、continue语句（不要求goto语句），另外for、switch语句作为可选项；
- 多维数组。另外结构作为可选项；
- 语言支持行注释与块注释，不要求支持编译预处理命令和多文件程序。

实验内容

- **实验内容：**完整可运行的自定义语言编译器
  - 实验一：词法语法分析器的设计与实现，生成抽象语法树。 建议使用词法语法分析程序生成工具如：LEX/FLEX ， YACC/BISON等专业工具完成。
  - 实验二：语义分析与符号表管理。对抽象语法树进行遍历，完成符号表的管理与相关属性计算。合理设计出符号表数据结构，如顺序表、HASH表等等，也可单张表格或多张表格，要求能动态展现符号表变化过程以便实验结果的检查。通过对符号表的管理实现语义分析。

实验内容

- 实验三：中间代码生成：对抽象语法树进行遍历，完成相关属性的计算，生成中间代码。中间代码的形式可以采用不同形式，但实验中要求定义自己的中间形式，建议采用四元组式。

  采用DAG，对中间代码进行局部优化。

- 实验四：目标代码生成：在前三个实验的基础上实现目标代码生成。

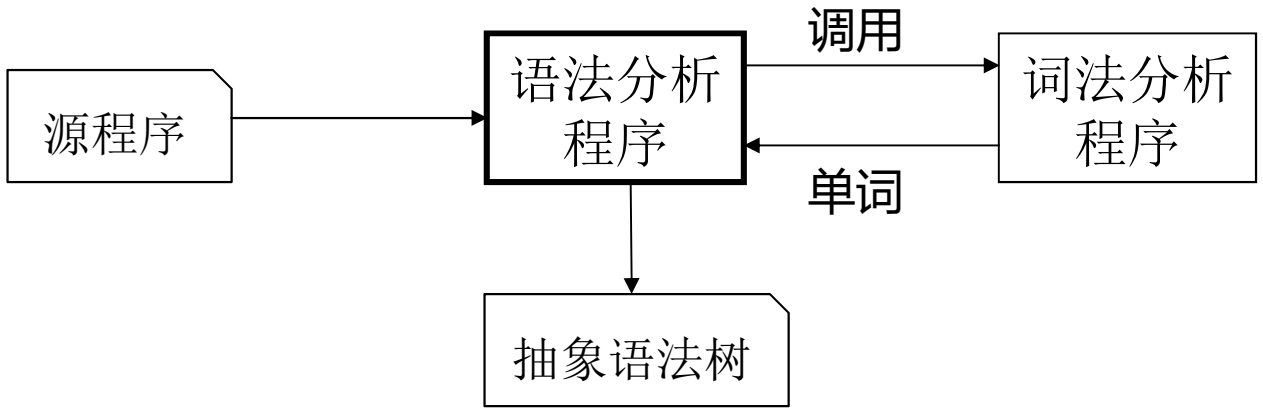
  也可以使用工具如LLVM来生成目标代码。

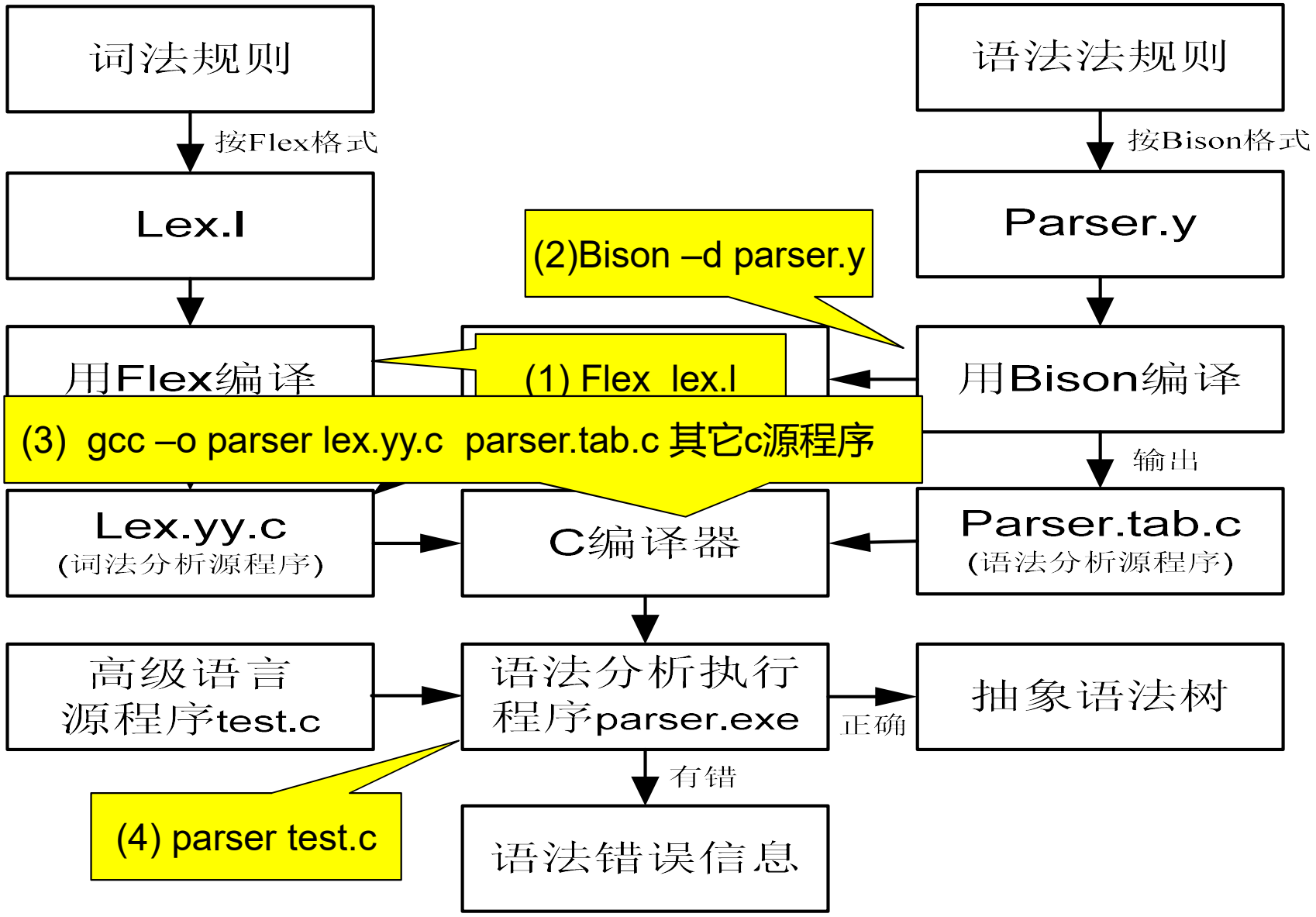
实验一 词法语法分析器的设计与实现

- 方法一：利用DFA识别原理进行词法分析器；采用递归下降分析法或LR分析法构造语法分析器，在语法分析过程中生成抽象语法树。
- 方法二：借助于工具实现：

  LEX/FLEX　　实现词法分析器

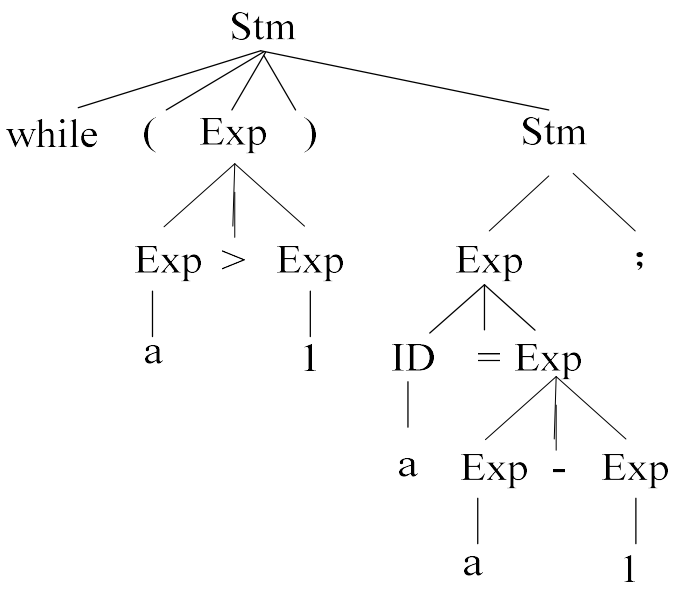
  YACC/BISON　　实现语法分析器

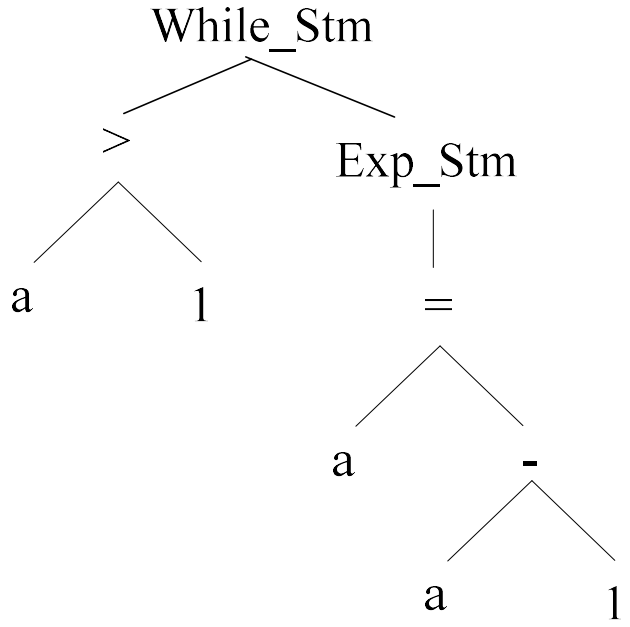



词法规则
按Flex格式
Lex.l
用Flex编译
Lex.yy.c
(词法分析源程序)
高级语言
源程序test.c
(4) parser test.c
语法法规则
按Bison格式
Parser.y
(2)Bison –d parser.y
用Bison编译
输出
Parser.tab.c
(语法分析源程序)
(1) Flex lex.l
(3) gcc –o parser lex.yy.c parser.tab.c 其它c源程序
C编译器
语法分析执行
程序parser.exe
正确
抽象语法树
有错
语法错误信息

抽象语法树：忽略程序语法成分细节，突出语法特征的语法树

例如： while (a>1) a=a-1;





语法推导树

抽象语法树

抽象语法树的物理结构可采用孩子表示法或二叉链表表示法

Lex/Flex源程序组成

```
%{
        声明部分
%}
        辅助定义部分
%%
        规则部分
%%
        用户子程序部分
```

三个部分都是可选的，没有用户子程序部分时，第2个%%可以省略

YACC/BISON源程序组成

%{

声明部分

%}

辅助定义

%%

规则部分

%%

用户子程序部分

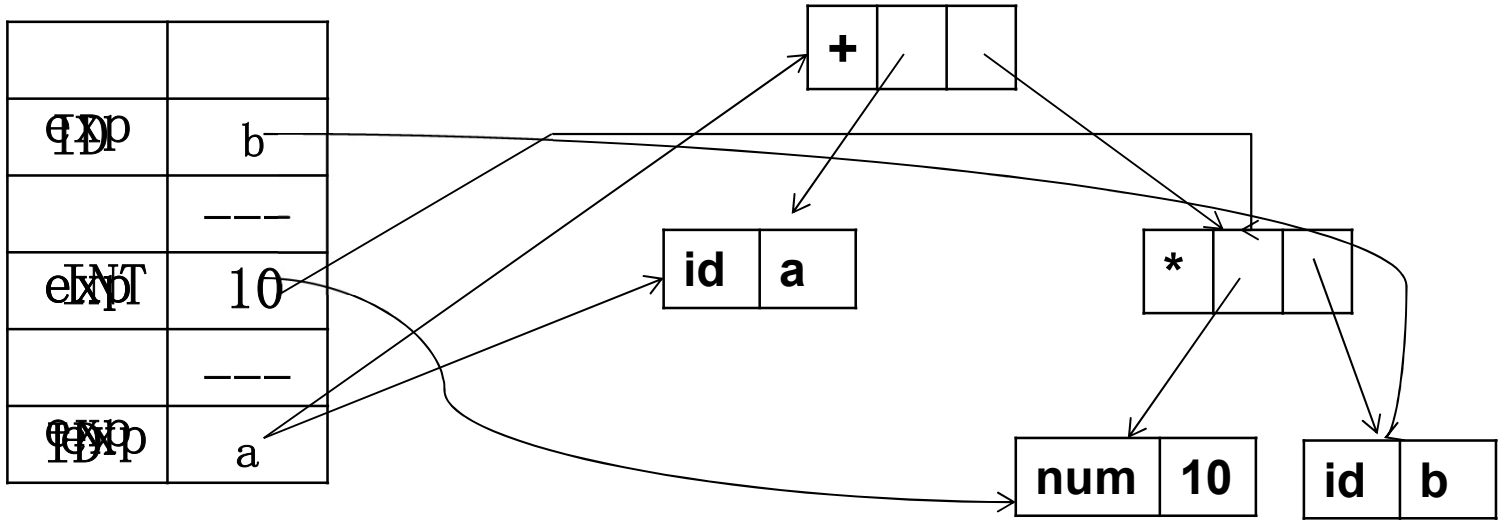
三个部分都是可选的，没有用户子程序时，第2个%%可以省略

FLEX和BISON联合使用，使用LR分析法，在归约过程中，建立抽象语法树，例如算术表达式文法：

Exp::=INT | ID | exp+exp | exp-exp | exp*exp | exp/exp
| (exp) | -exp

通过确定优先级，结合性来消除二义性

a + 10 * b



符号表b的入口

多行算术表达式的文法：

input::=input line | ε

line ::=exp ‘\n’ | ‘\n’

exp::=INT | ID | exp+exp | exp-exp | exp*exp | exp/exp

| (exp) | -exp

文件： Node.h

```
enum node_kind
{ID_NODE,INT_NODE,LPRP_NODE,PLUS_NODE,MINUS_NODE,STAR_NODE,DIV_NODE,UMINUS_NODE};

typedef struct Exp {
        enum node_kind kind;
        union {
                char type_id[33];          //由标识符生成的exp结点
                int type_int;              //由常数生成的exp结点
                struct {
                        struct Exp *pExp1;
                        struct Exp *pExp2;
                        }ptr;                //有运算生成的exp结点
        };
} *PEXP;
```

Flex文件：lex.l　　（后面的Bison文件名：exp.y）

```
%option yylineno
%{
#include "exp.tab.h"
#include "string.h"

typedef union {
        int  type_int;
        char type_id[32];
} YYLVAL;
#define YYSTYPE YYLVAL
/*yylval用来保存单词的属性，默认int，修改成YYLVAL*/
%}
id     [A-Za-z][A-Za-z0-9]*
int    [0-9]+
%%
{int}   {yylval.type_int=atoi(yytext); return INT;}
{id}    {strcpy(yylval.type_id,yytext);return ID;}
```

```
%%
{int}   {yylval.type_int=atoi(yytext); return INT;}
{id}    {strcpy(yylval.type_id,yytext);return ID;}
[+]     {return PLUS;}
[-]     {return MINUS;}
[*]     {return STAR;}
[/]     {return DIV;}
[(]     {return LP;}
[)]     {return RP;}
[ \t]   {;}
[\n]    {return yytext[0];}
.       {printf("error in line %d\n",yylineno);}
%%
int yywrap()
{
return 1;
}
```

YACC/BISON源程序

声明和辅组部分

```
%error-verbose
%locations
%{
#include "stdio.h "
#include "Node.h"
extern char *yytext;
extern FILE *yyin;
void display(struct Exp *,int);
%}
%union {    int type_int;
            char type_id[32];
            struct Exp *pExp;
};
%type  <pExp> line exp
%token <type_int> INT
%token <type_id> ID
%token LP RP  PLUS MINUS STAR DIV ASSIGNOP

%left PLUS MINUS
%left STAR DIV
%left UMINUS
%%
```

规则部分：

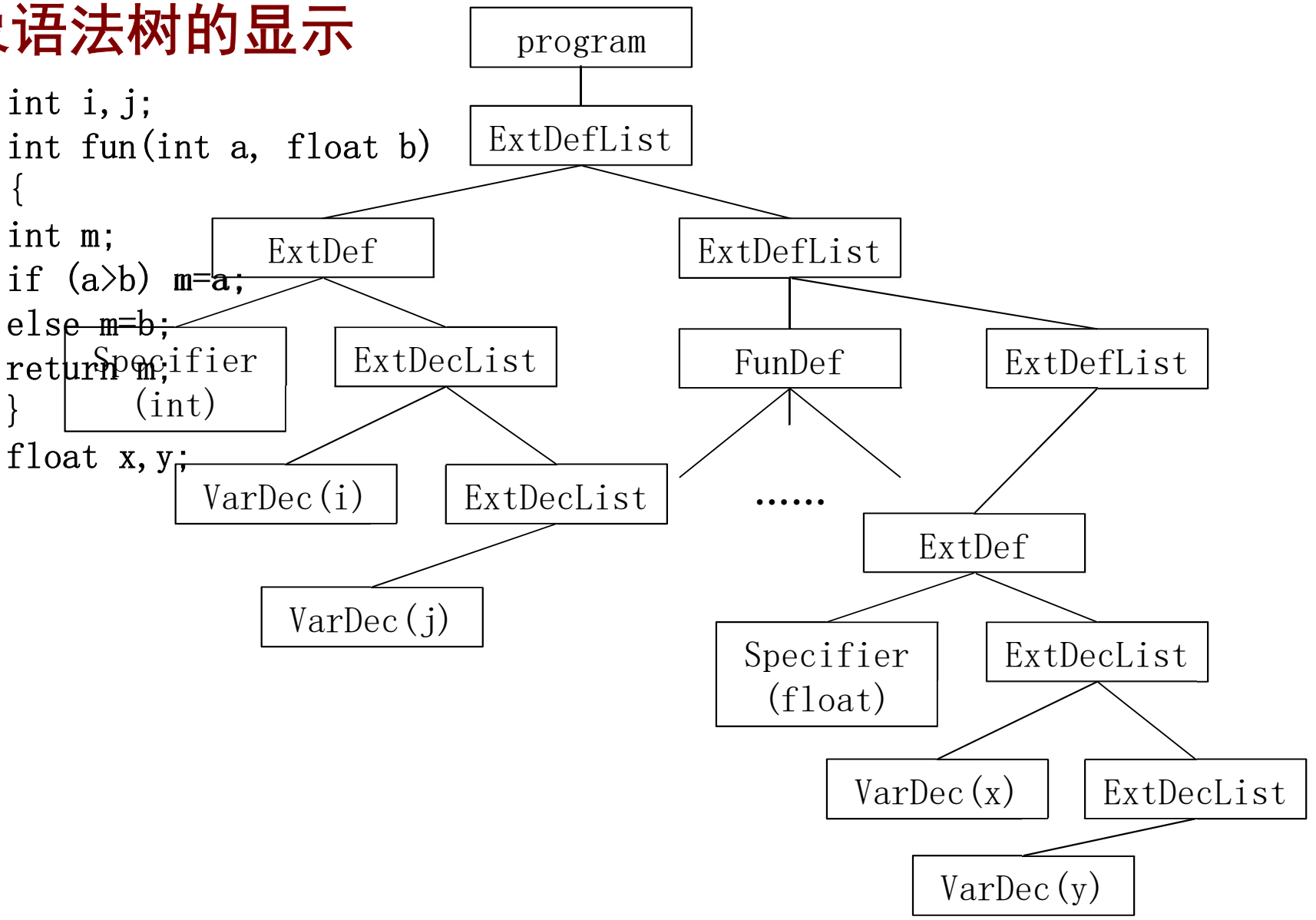
```
%%
input:
          | input line
          ;
line : '\n'     { ;}
          | exp '\n' { display($1,0);}                    /*显示语法树*/
          | error  '\n'  { printf("exp error!\n");} /*有语法错误时，跳过这行*
          ;
exp       : INT {$$=创建整数结点; 结点相关成员赋值;}
          | ID  {$$=创建标识符结点; 结点相关成员赋值;}
          | exp PLUS exp {$$=创建加运算结点; $$->kind=PLUS_NODE;
                          $$->ptr.pExp1=$1;$$->ptr.pExp2=$3;}
        ..................
        | MINUS exp  %prec UMINUS {$$=创建单目符号运算结点;结点相关成员赋值;}
        ;
        /*以上exp的规则的语义动作生成抽象语法树*/
%%
```

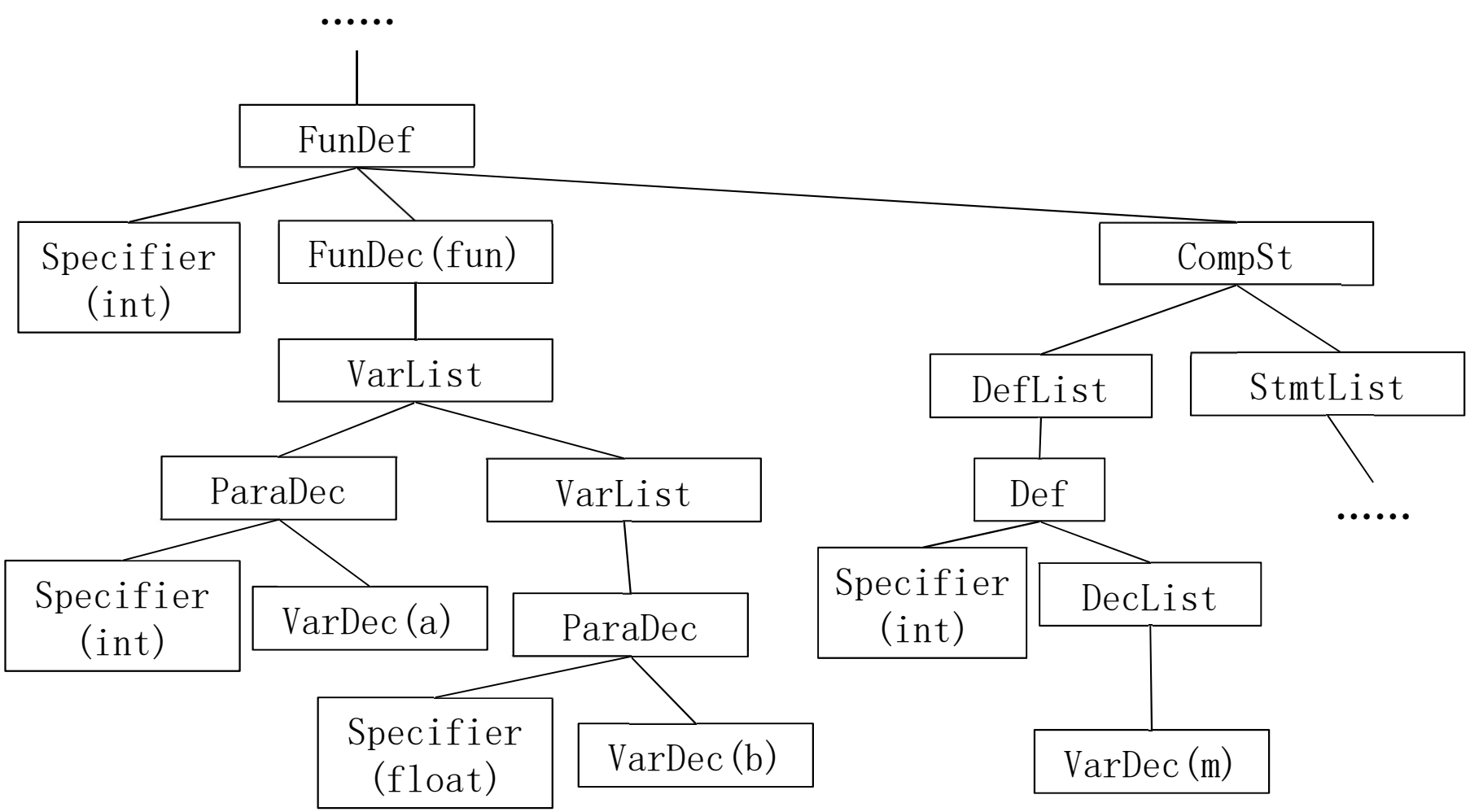
用户子程序部分部分：

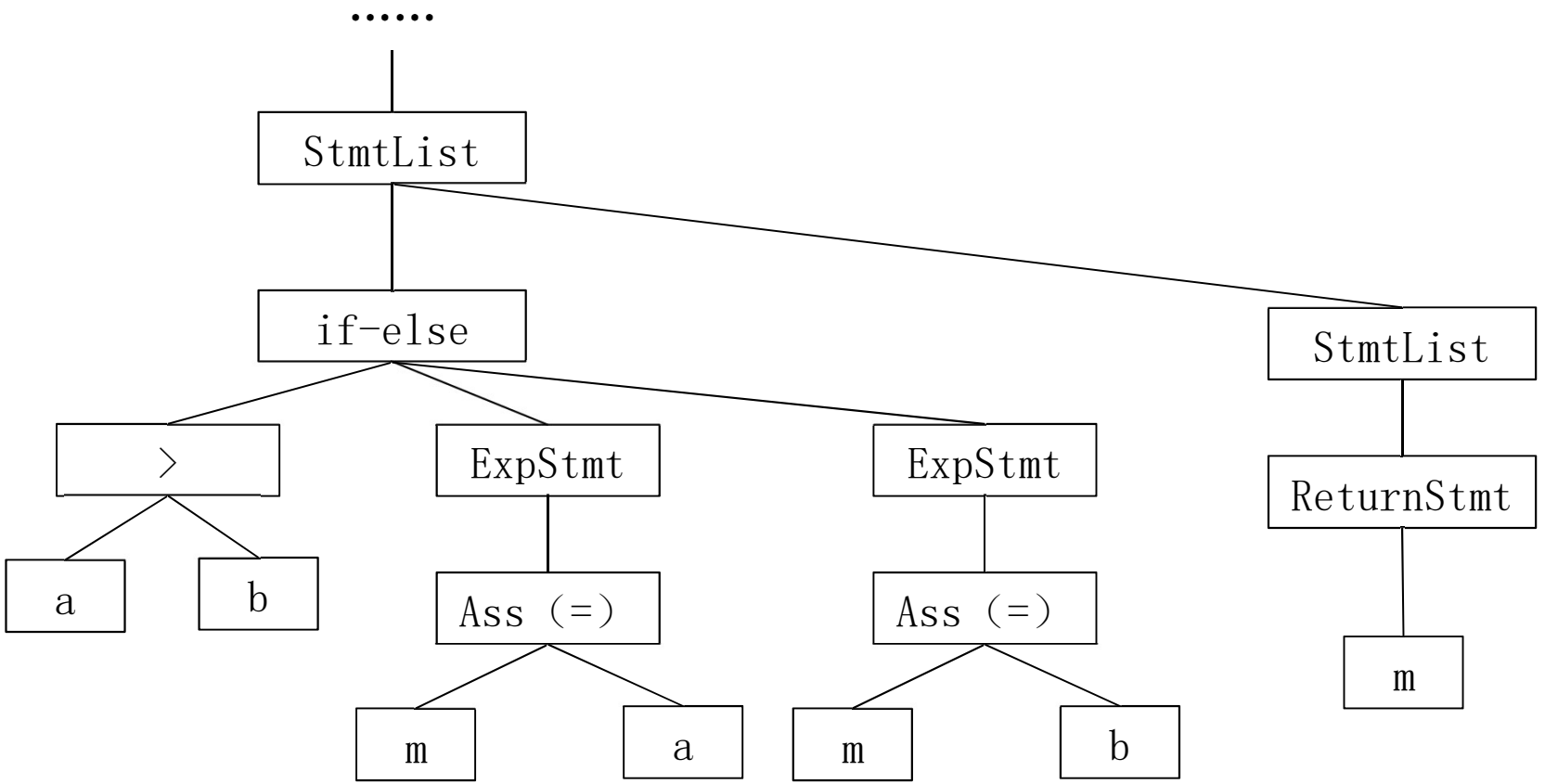
```
%%

int main(int argc, char *argv[]){
        yyin=fopen(argv[1],"r");
        if (!yyin) return;
        yyparse();
        return 0;
        }

yyerror(char *s){
   printf("%s   %s \n",s,yytext);

 }
```

抽象语法树的显示

```
int i,j;
int fun(int a, float b)
{
int m;
if (a>b) m=a;
else m=b;
return m;
}
float x,y;
```




.....
FunDef
Specifier (int)
FunDec(fun)
CompSt
VarList
DefList
StmtList
ParaDec
VarList
Def
.....
Specifier (int)
VarDec(a)
ParaDec
Specifier (int)
DecList
Specifier (float)
VarDec(b)
VarDec(m)


.....
StmtList
if-else
StmtList
>
ExpStmt
ExpStmt
ReturnStmt
a
b
Ass (=)
Ass (=)
m
m
a
m
b

抽象语法树的显示

```
int i,j;
int fun(int a, float b)
{
int m;
if (a>b) m=a;
else m=b;
return m;
}
float x,y;
```

实验一的检查验收

- 自行准备好测试用例。
- 词法分析：正确识别出源程序中的单词，以二元组的形式显示。以及对不能识别的符号报错。
- 语法分析：（1）正确显示出测试用代码的抽象语法树，要求根据显示的形式能还原出源程序的代码（不包含注释）；（2）报错功能，能正确给出错误性质和位置，并有容错的功能（能够1次显示多个语法错误）。
- 回答老师的提问。
- 评分依据：（1）完成的时间；（2）完成的质量。

实验二 符号表的管理与语义检查

对实验一生成的抽象语法树进行先根遍历，完成：

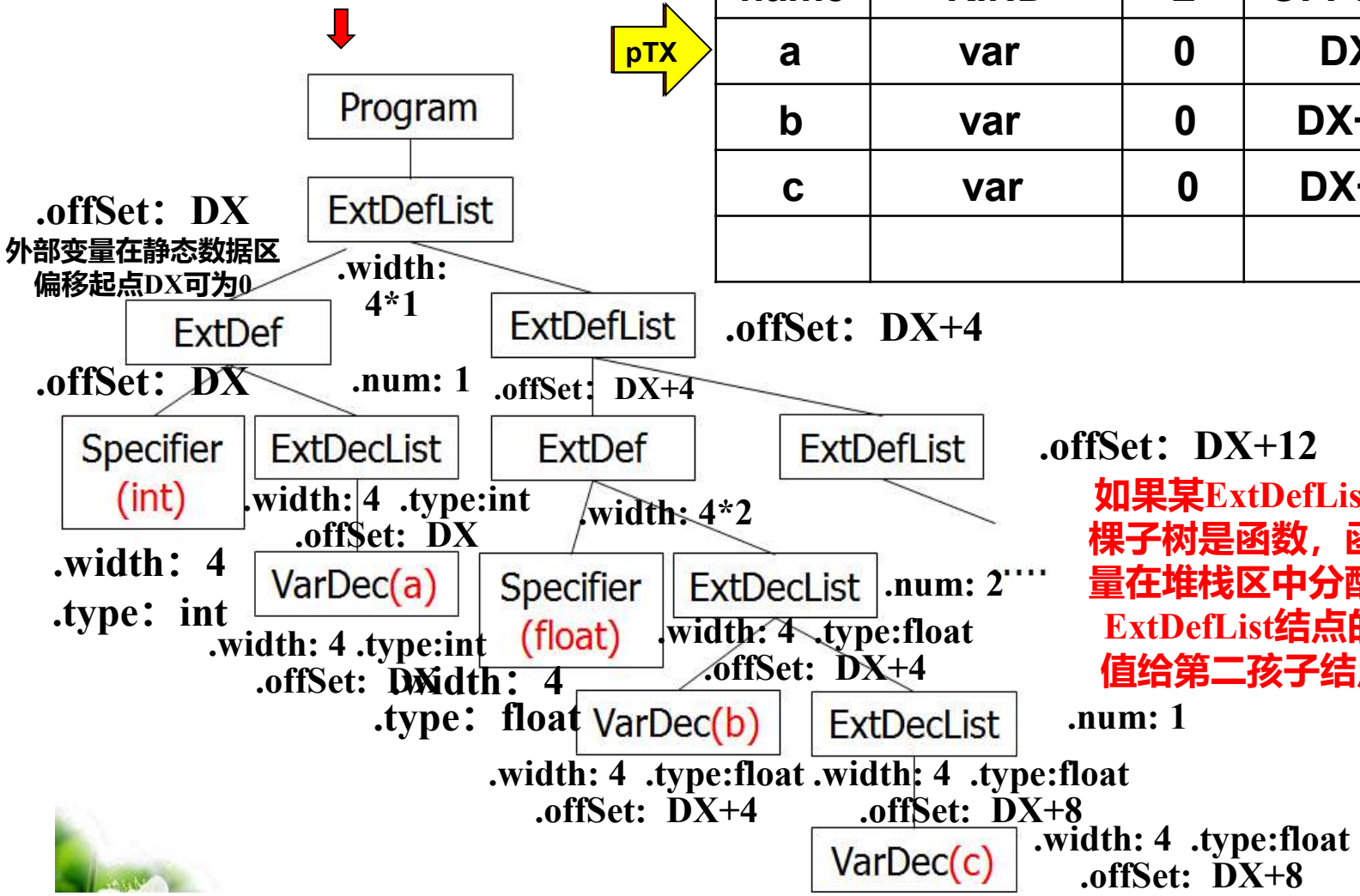
- 符号表的管理。可选择采用顺序表，HASH表，十字链表，多表等形式。遇到说明语句填写符号表并做出相应的语义检查；遇到语句中的符号引用，查找符号表并进行语义检查。
- 类型检查
- 名字的作用域分析
- 控制流检查（break，continue等必须出现在合法的位置
- 。。。

(1) 外部声明

LEV: 0

name	KIND	L	OFFSET	TYPE…
a	var	0	DX	int
b	var	0	DX+4	float
c	var	0	DX+8	float

pTX



如果某ExtDefList结点的第一个棵子树是函数，函数中的局部变量在堆栈区中分配单元，则将该ExtDefList结点的offSet直接赋值给第二孩子结点的offset属性

LEV: 0

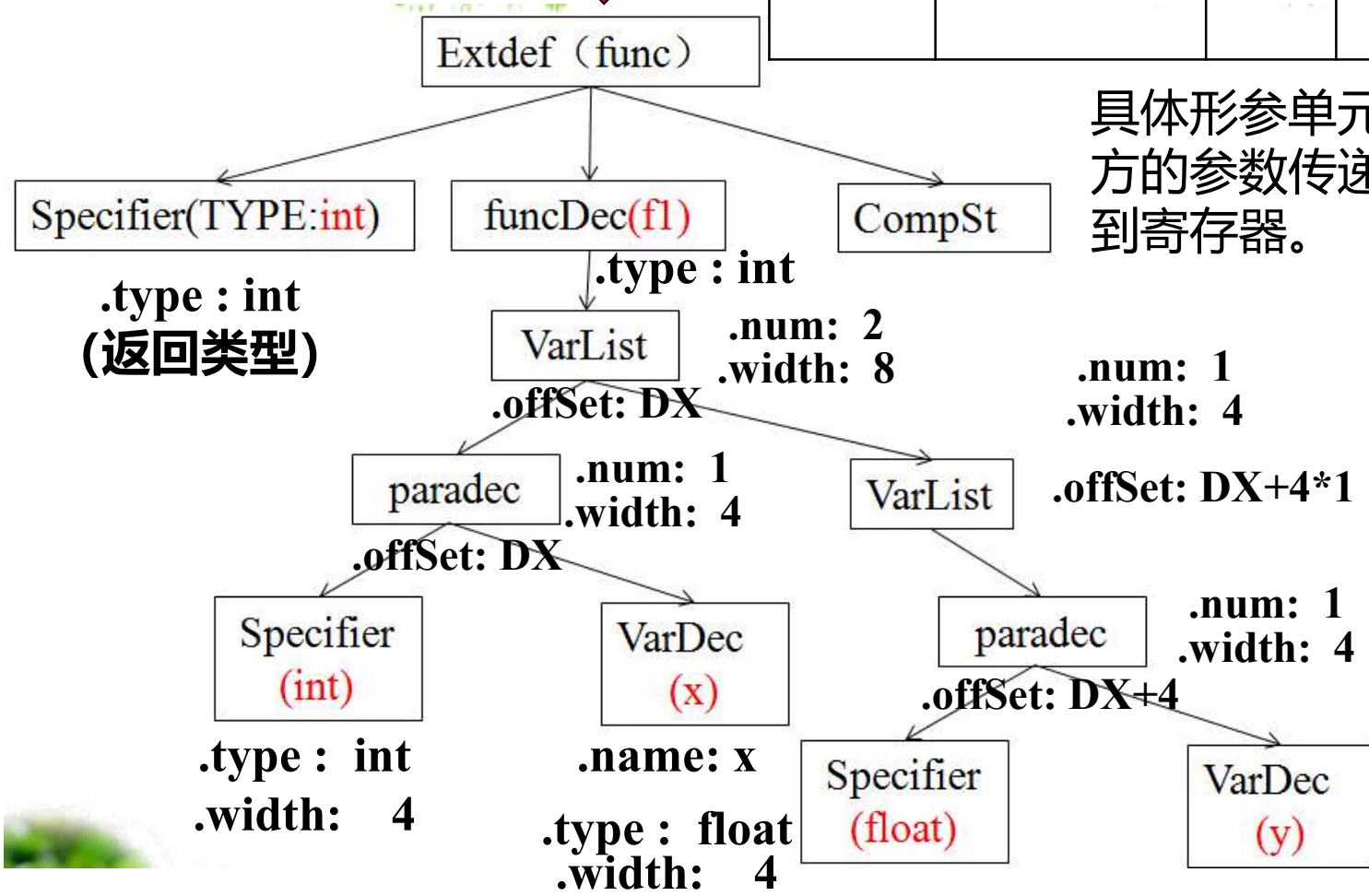
(2) 函数声明

pTx=>TX1

pTX

TX1

name	KIND	L	OFFSET	TYPE…
f1	function	0	函数空间大小	返回类型等
x	Var	1	DX	int
y	Var	1	DX+4	float



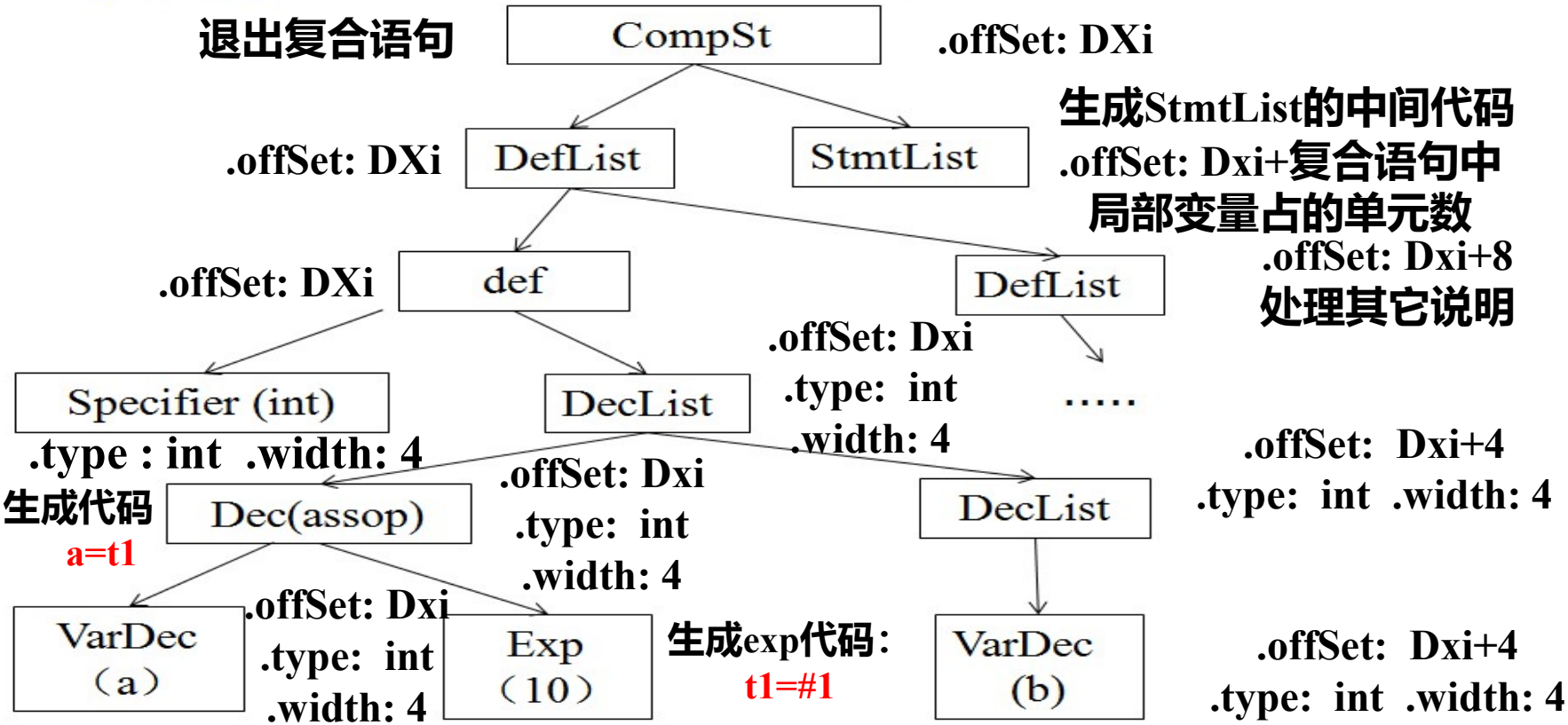
具体形参单元分配依赖于调用双方的参数传递的约定，通常会用到寄存器。

(3) 复合语句

LEV: i-1

pTX

name	KIND	L	OFFSET	TYPE…
a	var	i	DXi	int
b	var	i	Dxi+4	int

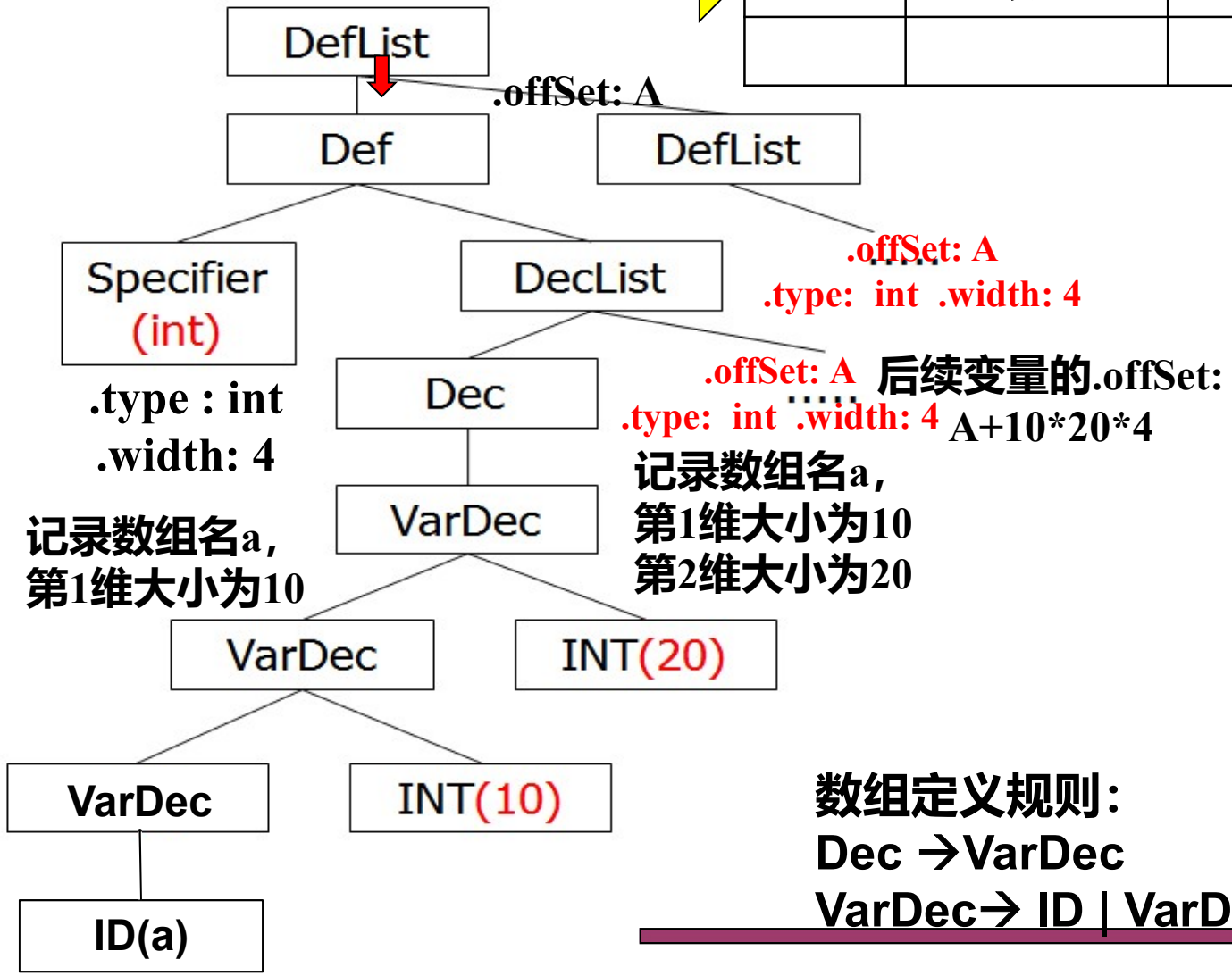


LEV: i

(4) 数组说明

name	KIND	L	OFFSET	OTHER
a	数组	i	DXi	内情向量

pTX



维数 2	元素类型 int	
首地址 A	不变部分 C	
0	9	10
0	19	20
下界	上界，界差	

数组定义规则：

Dec →VarDec

VarDec→ ID | VarDec LP INT RP

实验二的检查验收

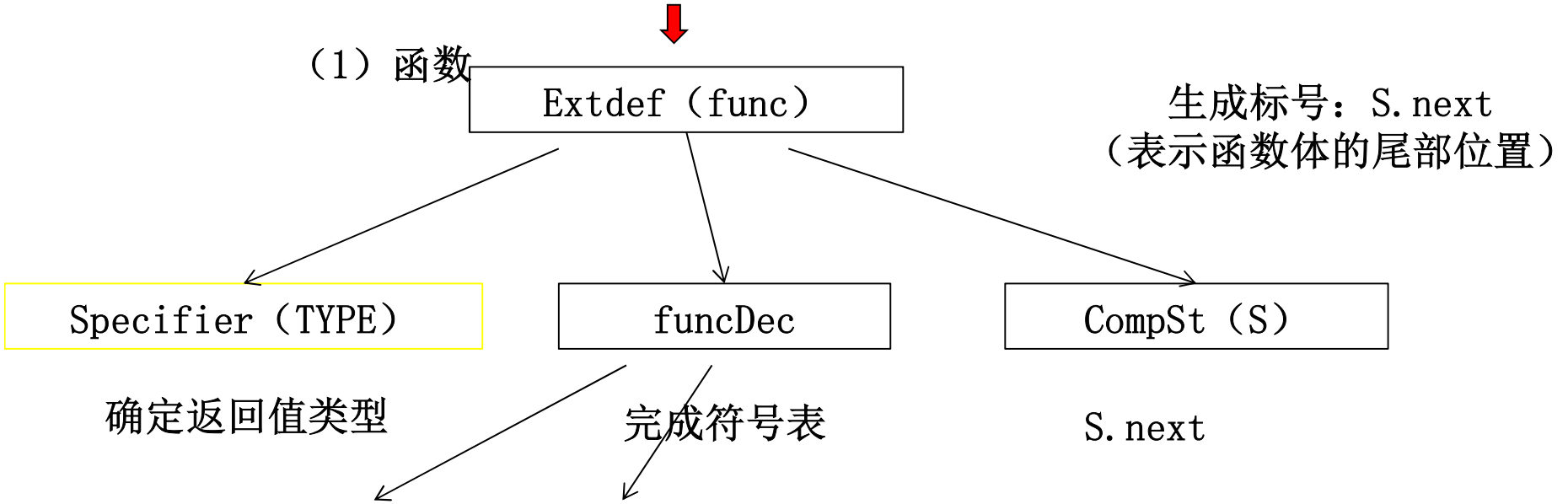
- 自行准备好若干个测试用例，测试用例中包含的语义错误类别可以参考实验指导教程和实验指导教程中参考教材[2]。
- 回答老师的提问。可根据老师的要求，在程序的合适位置加上符号表的显示。
- 评分依据：（1）完成的时间；（2）完成的质量（主要看能检查出多少静态语义错误）。

实验三 中间代码生成

当无静态语义错误时，生成中间代码。

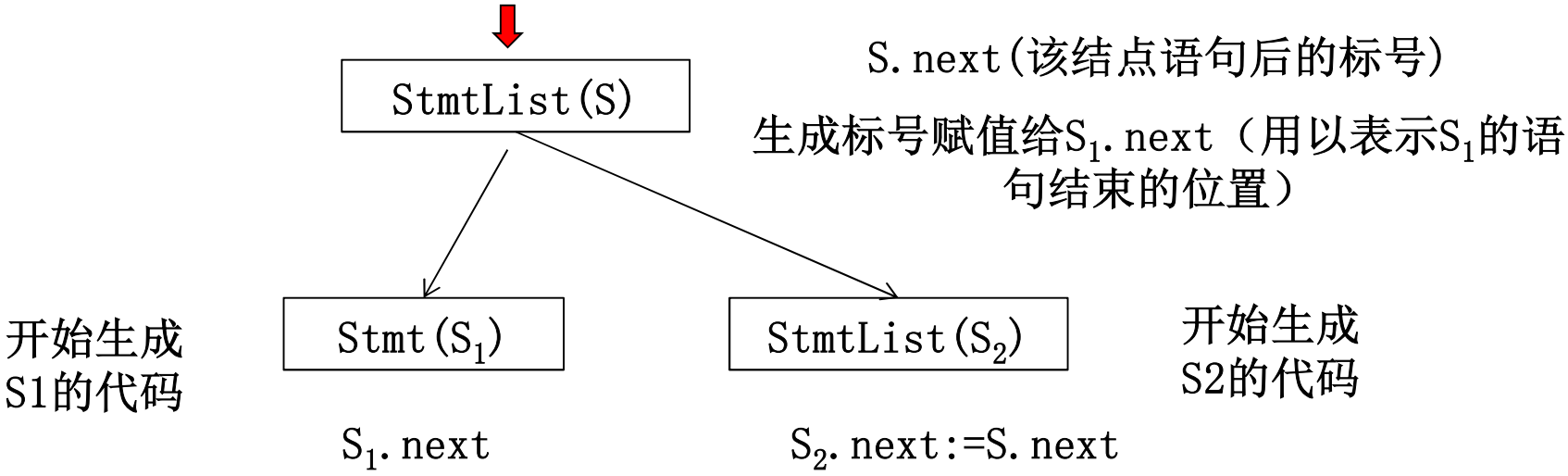
结合符号表，对抽象语法树进行先根遍历，在遍历过程中，需要完成继承属性和综合属性的计算，同时生成四元组式的中间代码（每一个四元组式内部实际上是一个结构型的数据，这样在实验四中就能方便地完成目标代码的生成）。

2. 中间代码(参照P216翻译模式理解属性计算次序)



该结点访问结束时
Func.code=S.code||gen(S.next’:’)
填写函数入口（也可以在前面的符号表处理时，用标号的形式给出函数入口）
，函数局部变量等需要的空间大小

（2） 语句序列的翻译



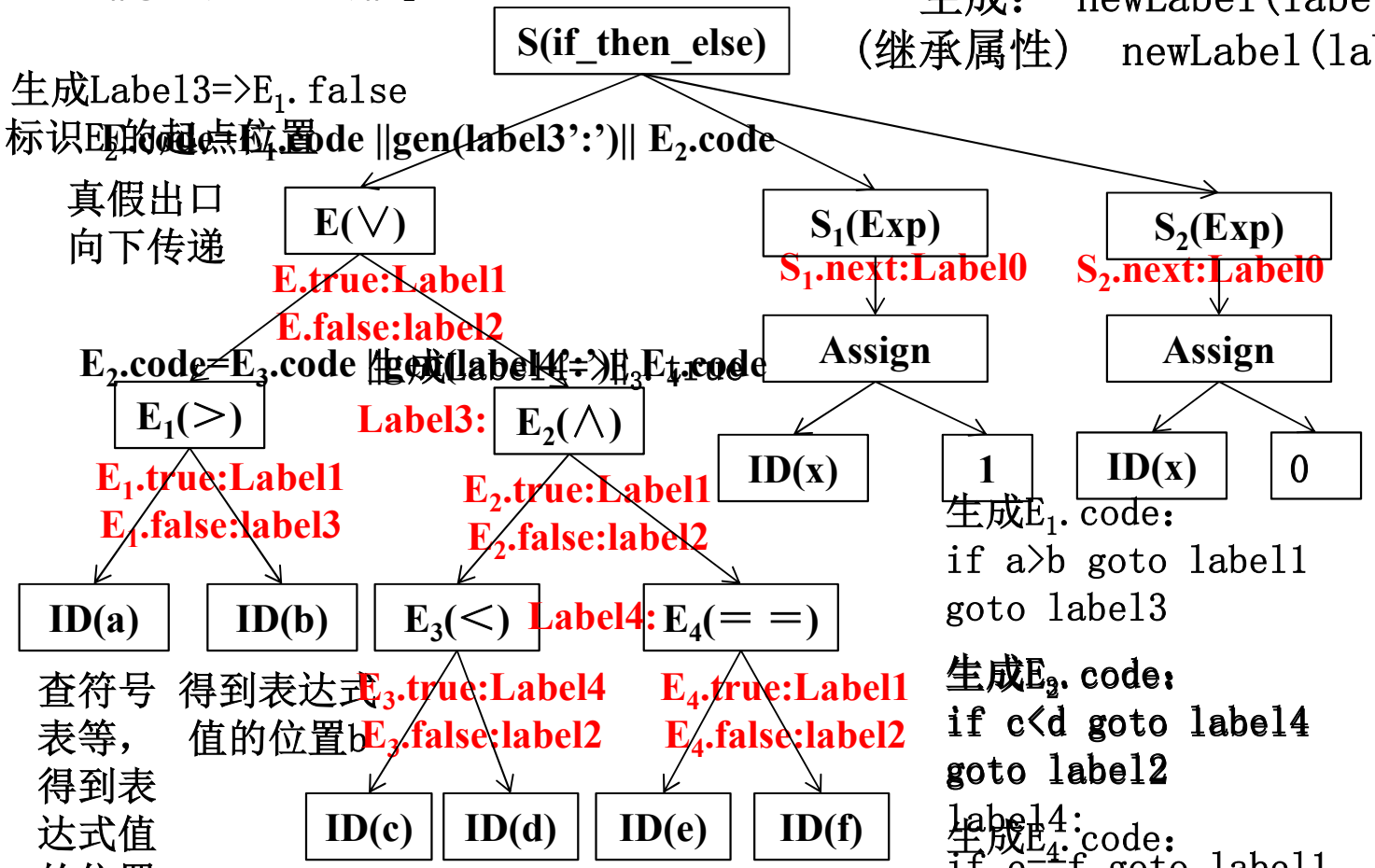
当各子树遍历完成后：

S.code= S1.code ||gen(S1.next ':') || S2.code

S.code=E.code || gen(label1':')
||S_1.code||gen(goto label0)
|| gen(label2':')||S_2.code

S.next： label0

生成： newLabel(label11)赋值给E.true
(继承属性) newLabel(label12)赋值给E.false



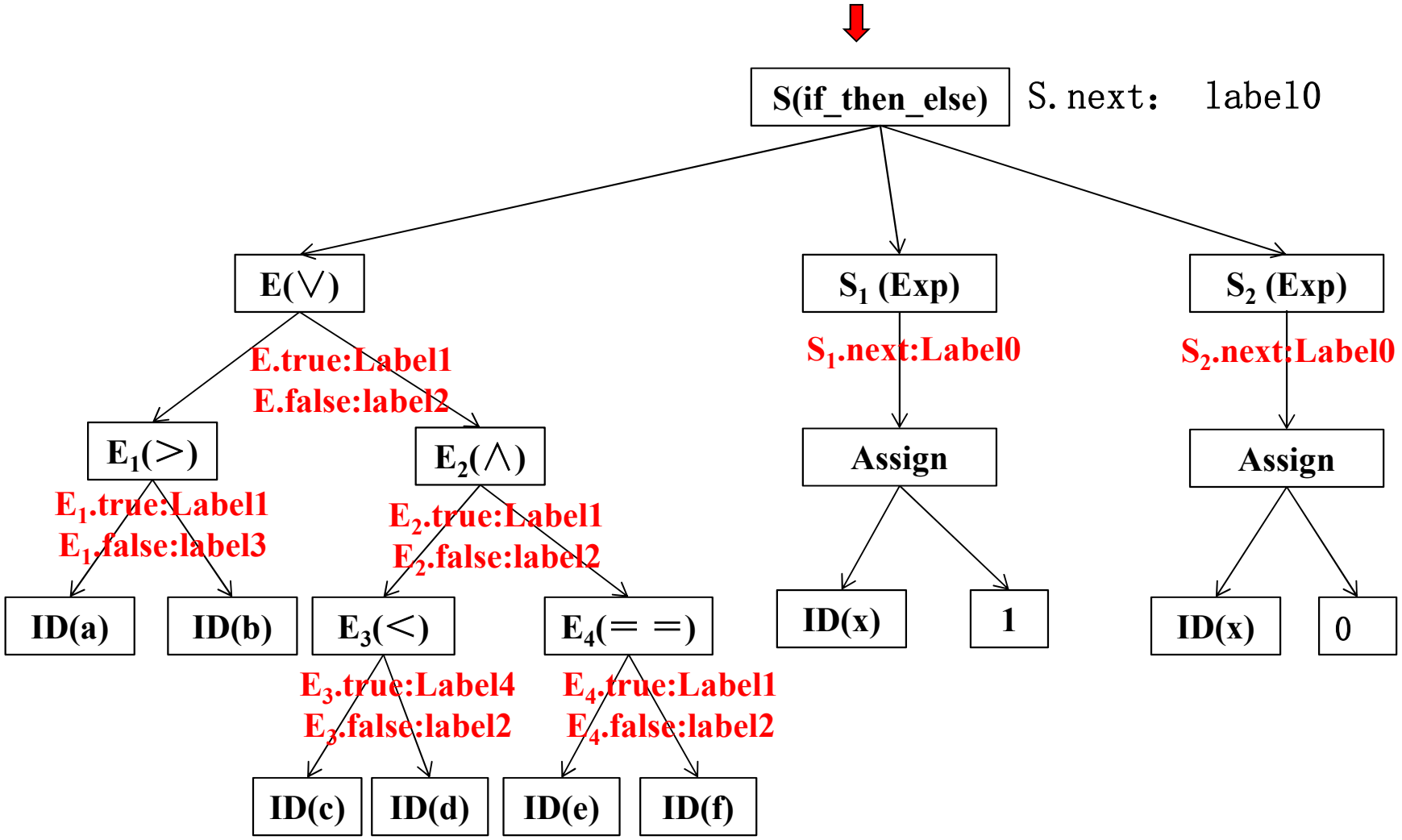
生成E_1.code:
if a>b goto label1
goto label3

生成E_2.code:
if c<d goto label4
goto label2
label4:
生成E_4.code:
if e==f goto label1
goto label2

S.code:
if a>b goto label1
goto label3
Label3:
if c<d goto label4
goto label2
label4:
if e=f goto label1
goto label2
label1:
S_1.code:
t1=1
x=t1
goto label0

label2:
t2=0
x=t2

label0:


S(if_then_else)
S.next: label0
E(∨)
S1 (Exp)
S2 (Exp)
E.true:Label1
E.false:label2
S1.next:Label0
S2.next:Label0
E1(>)
E2(∧)
Assign
Assign
E1.true:Label1
E1.false:label3
E2.true:Label1
E2.false:label2
ID(a)
ID(b)
E3(<)
E4(= =)
ID(x)
1
ID(x)
0
E3.true:Label4
E3.false:label2
E4.true:Label1
E4.false:label2
ID(c)
ID(d)
ID(e)
ID(f)

中间代码生成中的换名问题

```
int  func(){
     int  a;
       a=10;
      {
         int a;
          a=20;
          }
return a;
}
```


```
FUNCTION func:
 a=10
 a=20
RETURN a
```

显然是错误的中间代码，原因是在高级语言通过复合语句区分不同的作用域，而在中间代码却没有这个功能，所以在符号表的建立时，考虑到后续的实验3、4，需要给变量进行换名。

```
int  func(){
   int  a;
   a=10;
   {
     int a;
     a=20;
    }
return a;
}
```

pTX

TX2

name	KIND	L	OFFSET	ALIAS
a	var	1	DX	v_i
a	var	1	DX+4	V_{i+1}

FUNCTION func:

v_i=10

V_{i+1}=20

RETURN V_i

实验三的检查验收

- 自行准备好若干个测试用例，生成包括函数调用、递归、条件语句、循环语句、break、continue、数组、自增与自减等的中间代码。

  注意：（1）一个测试程序不宜包含太多内容，以免引起检查的不方便。（2）不需要显示前面实验一、二的内容，直接输出中间代码。（3）能分别显示优化前和优化后的中间代码。

- 回答老师的提问。
- 评分依据：（1）完成的时间；（2）完成的质量。

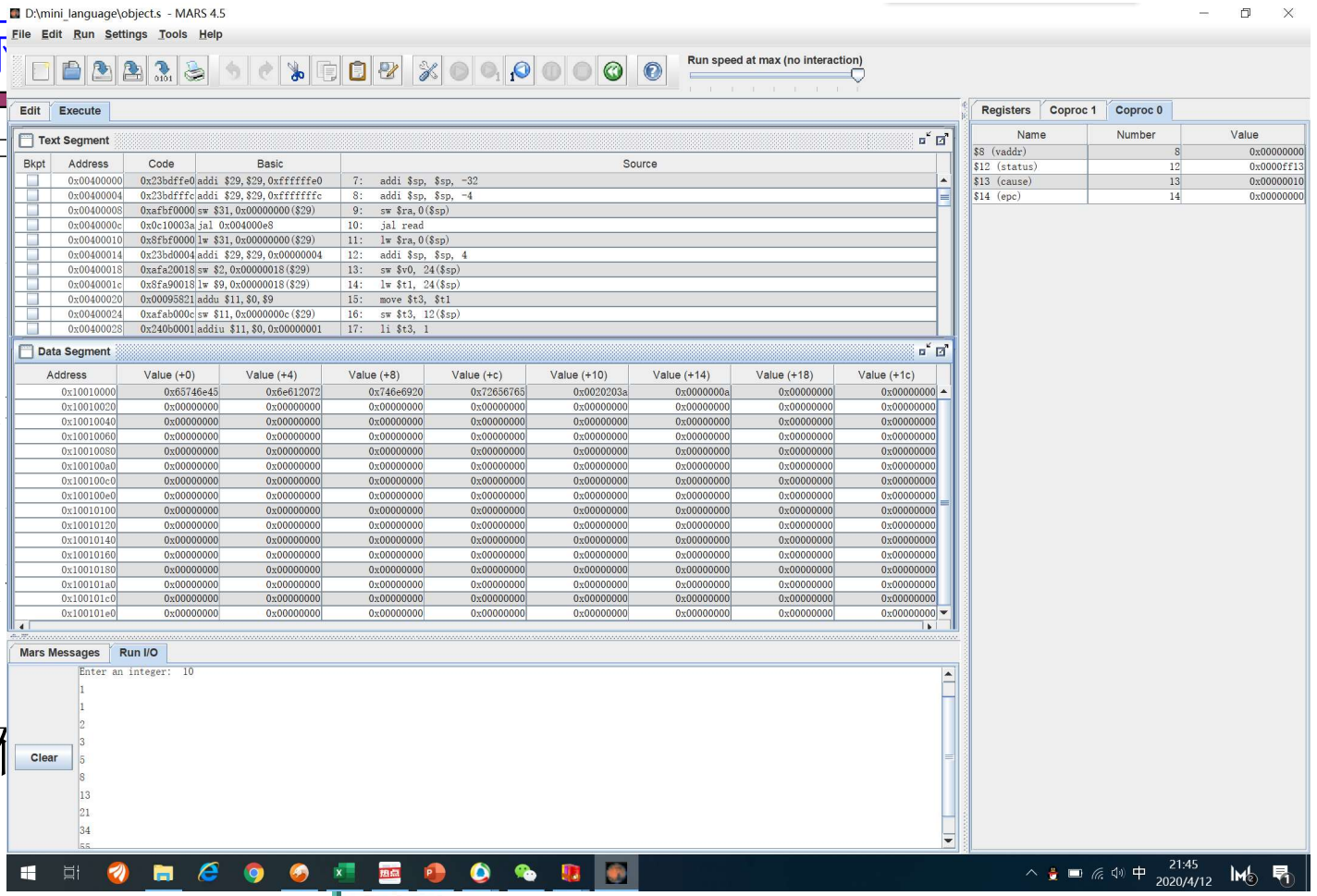
实验四 目标

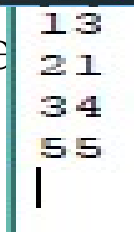
将实验三的中

- 选择目标代码
- 选择指令集（
- 寄存器分配算
- 目标代码生成
- 或使用LLVM直

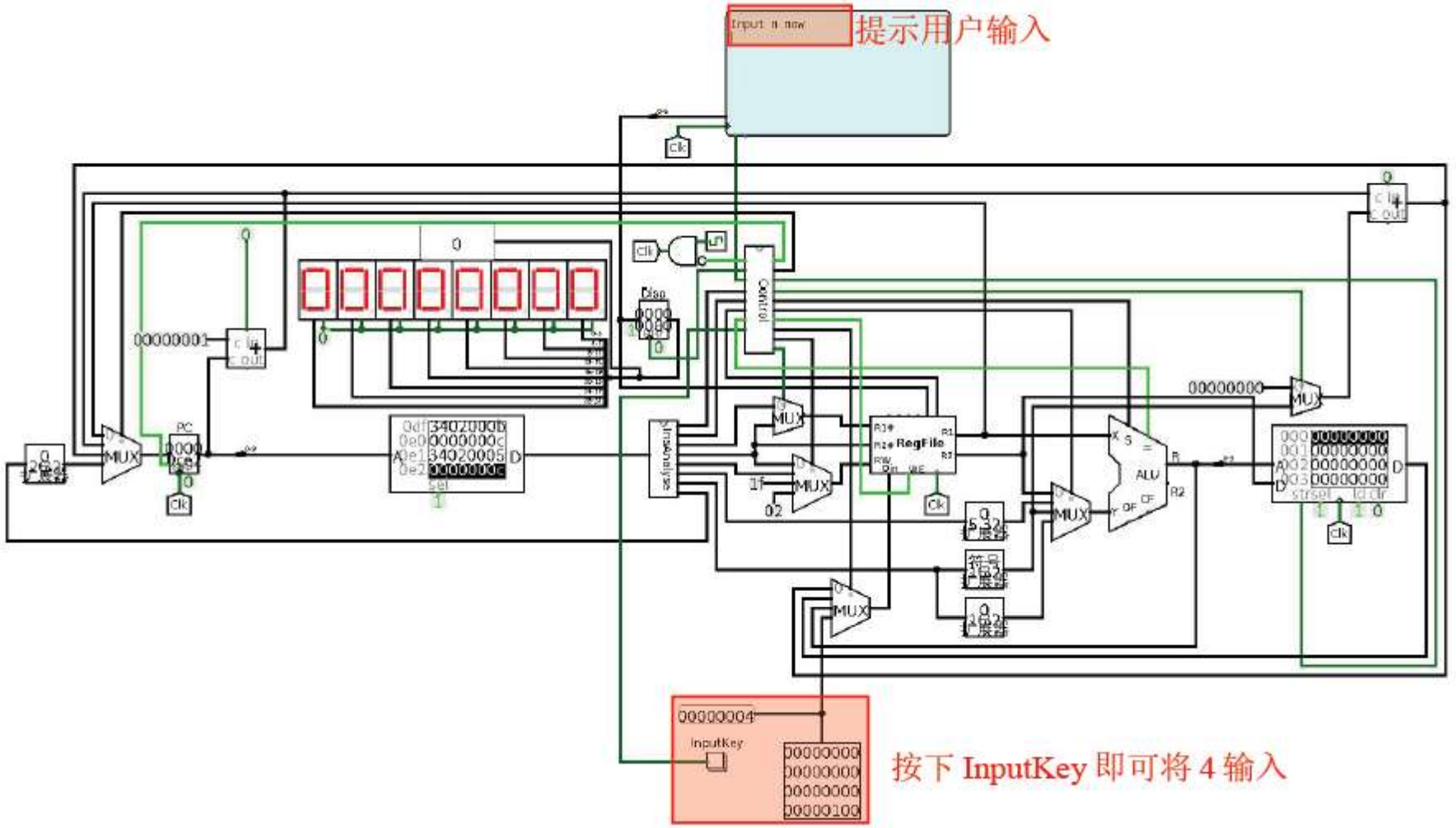
MIPS的执行环

1. QTSPIM
2. MARS （MIPS Assemble
3. 组成原理构建的环境





与组成原理实验对接


提示用户输入
Input a new
Clk
Control
RegFile
ALU
MUX
InputKey
00000004
00000000
00000000
00000000
00000100
按下 InputKey 即可将 4 输入

实验四的检查验收

- 自行准备好若干个测试用例，包括函数调用、递归、条件语句、循环语句、break、continue、数组、自增与自减等。

  注意： 不需要显示前面实验一、二、三的内容，直接对源程序进行编译，生成目标代码并执行。
- 回答老师的提问。
- 评分依据： （1）完成的时间；（2）完成的质量。