XYZ COMPILER PROJECT FINAL REPORT

Team 8

5060379026	何轩
5060379019	陈啸邑
5060379021	耿乾坤
5060379069	毛松亮
5060379002	刘辉

Contents

Our work

➢ Preface

The work of building XYZ Compiler is divided into sequential four steps:

- ★ Lexical Analysis -> Token
- ★ Syntax Analysis -> AST
- ★ Semantic Analysis -> Symbol table
- ★ Code generation -> 3AC

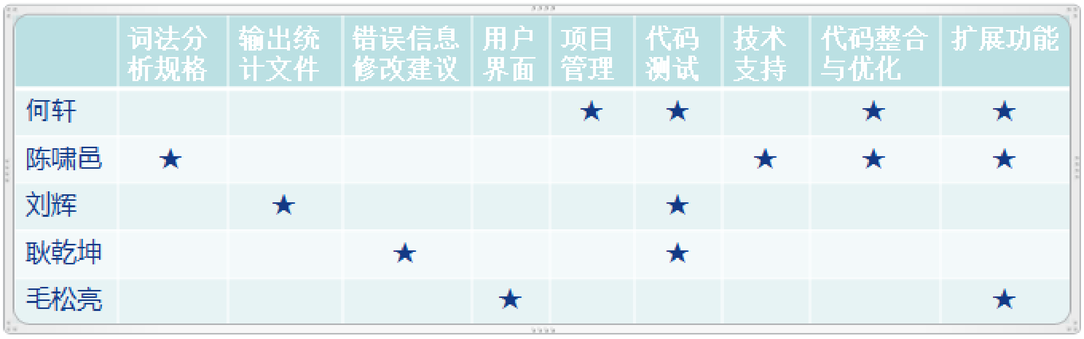
The **techniques used** are JAVACC, JJTree, Symbol Table and some basic data structures and algorithms such as Tree and Recursion.

➢ Lexical Analysis Part :

Description:

First of all, we defined the lexical analysis rules in the jj file. After this, we calculate each kind of defined key words in the input file and store them in Hashtable which is used to count and print the final results. We also show the lexical errors and suggestions to the users. A friendly GUI was also implemented. Please see Appendix A for more details.

Duty:



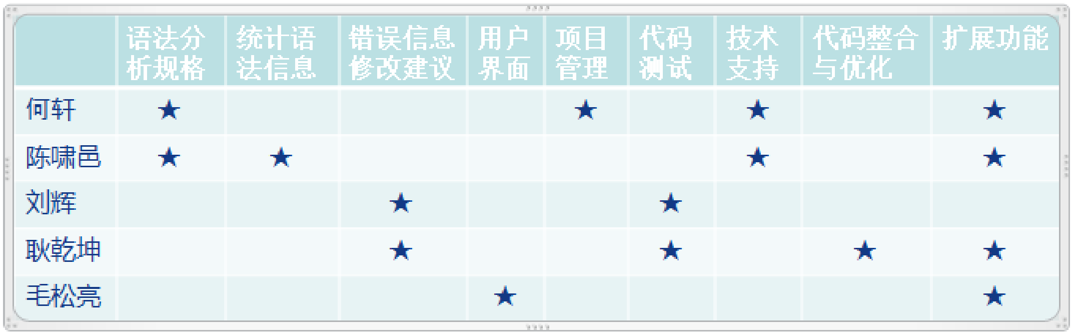
	词法分析规格	输出统计文件	错误信息修改建议	用户界面	项目管理	代码测试	技术支持	代码整合与优化	扩展功能
何轩					★	★		★	★
陈啸邑	★						★	★	★
刘辉		★				★			
耿乾坤			★			★			
毛松亮				★					★

➢ Syntax Analysis Part:

Description:

In this part, we build the Abstract Syntax Tree (AST) with jjtree in jjt file. After this, we statics some syntax information and make AST visualization. We also show the syntax errors and suggestions to the users. Please see Appendix B, C, D for more details.

Duty:



	语法分析规格	统计语法信息	错误信息修改建议	用户界面	项目管理	代码测试	技术支持	代码整合与优化	扩展功能
何轩	★				★		★		★
陈啸邑	★	★					★		★
刘辉			★			★			
耿乾坤			★			★		★	★
毛松亮				★					★

➢ Semantic Analysis Part:

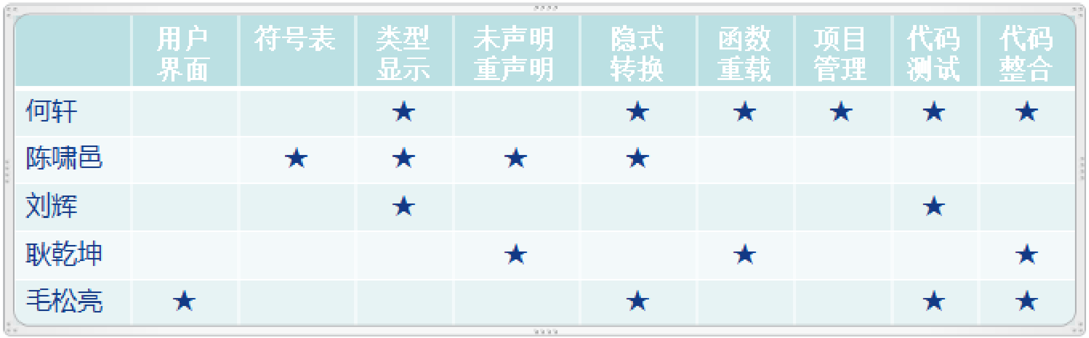
Description:

In this part, we deal with some semantic Verification:

- ★ Undeclared & Redeclared Identifier
- ★ Type Mismatch
- ★ Implicit Conversion
- ★ Function Overloading

Please see Appendix C, D and E for more details.

Duty:



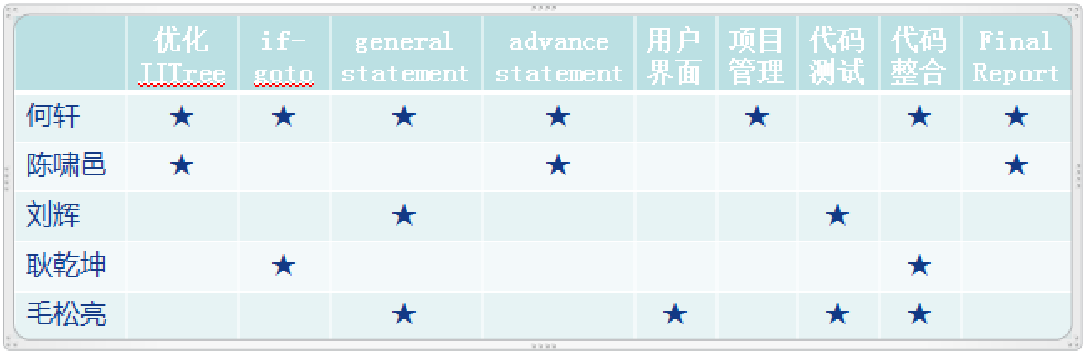
	用户界面	符号表	类型显示	未声明重声明	隐式转换	函数重载	项目管理	代码测试	代码整合
何轩			★		★	★	★	★	★
陈啸邑		★	★	★	★				
刘辉			★					★	
耿乾坤				★		★			★
毛松亮	★				★			★	★

➢ Code Generation Part:

Description:

In this part, we convert the AST into Three Address Code (3AC). The main emphasis of our work is focused on the “if else do while” structure and Assignment statement. Please See Appendix C, D for more details.

Duty:



	优化 JJTree	if-goto	general statement	advance statement	用户界面	项目管理	代码测试	代码整合	Final Report
何轩	★	★	★	★		★		★	★
陈啸邑	★			★					★
刘辉			★				★		
耿乾坤		★						★	
毛松亮			★		★		★	★	

Experiences & Lessons

In this project, first we deeply understand the complete procedure and basic concept of compiler. We also learn some applications of compiling techniques such as JavaCC. Finally, we gain much precious programming experience through a lot of practice. Thanks a lot to this project.

Potential Use

- The compiler techniques can be used in the test of software to analysis the lexical and syntax.
- The lexical and syntax analysis techniques can be used in the searching Engine.
- The compiler techniques can be used to optimize code of software.

Suggestions

- I think it may be better if the professor gives some instructions or have a talk about the part before we start doing it.
- After each phase, the professor should talk about all the implementations and have an analysis about them.

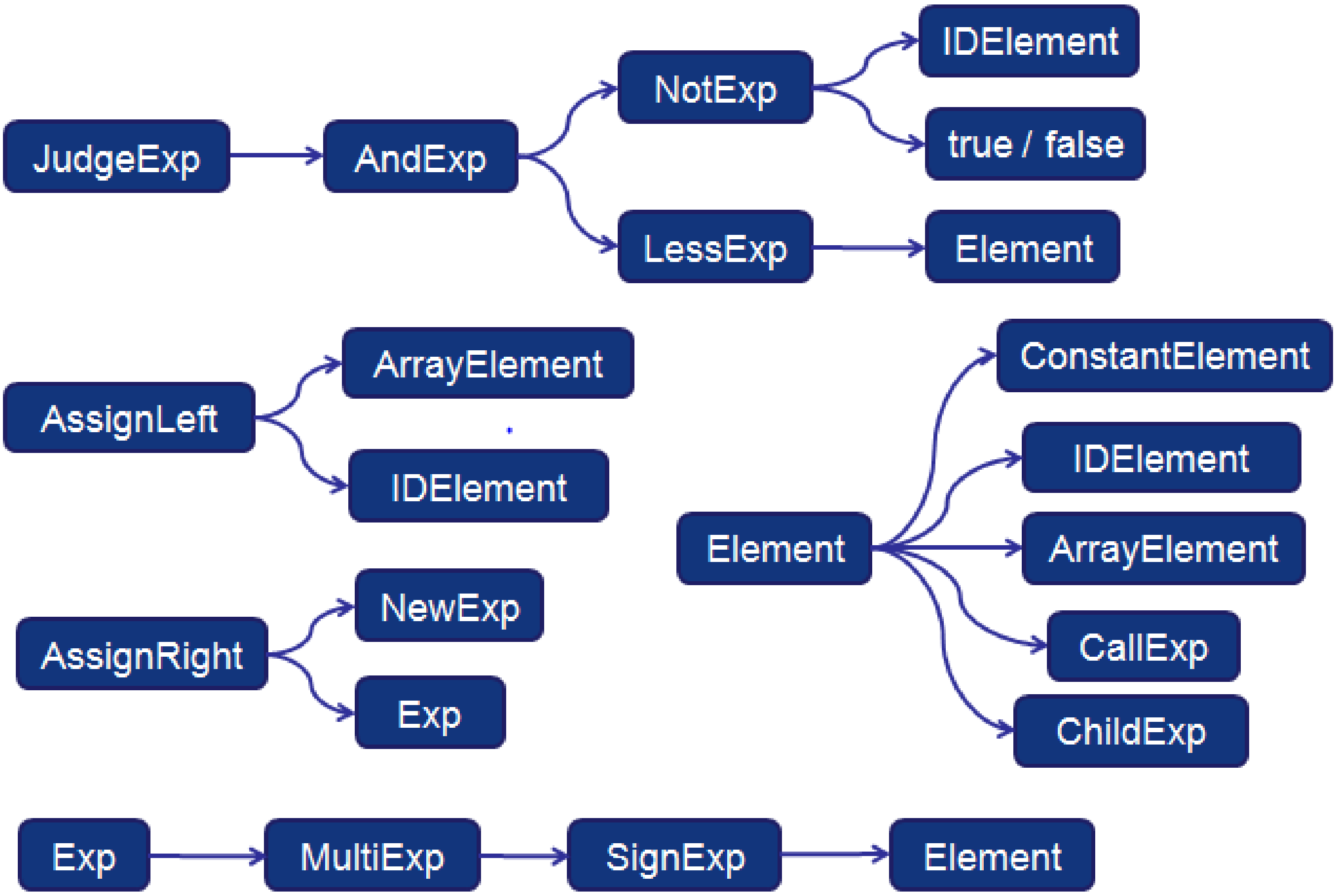
Appendix A. Tokens Definition

```
/* WHITE SPACE */
SKIP :
{
    " " | "\t" | "\n" | "\r" | "\f"
}

/* OPERATIONS */
TOKEN :
{
    <NOT:"!"> | <ASSIGN:"="> | <AND:"&&"> | <LT:"<"> | <PLUS:"+"> |
    <MINUS:"-"> | <MUL:"*"> |
    <OPER:(<NOT>|<ASSIGN>|<AND>|<PLUS>|<LT>|<MINUS>|<MUL>)>
}

/* KEY WORDS */
TOKEN :
{
    <PUBLIC:"public"> | <CLASS:"class"> | <EXTENDS:"extends"> |
    <STATIC:"static"> | <VOID:"void"> | <MAIN:"main"> | <STRING:"String">
    | <IF:"if"> | <ELSE:"else"> | <DO:"do"> | <WHILE:"while"> | <NEW:"new">
    | <LENGTH:"length"> | <PRINT:"System.out.println"> | <INT:"int"> |
    <BOOLEAN:"boolean"> | <TRUE:"true"> | <FALSE:"false"> | <THIS:"this">
    | <RETURN:"return"> | <DOUBLE:"double">|
    <KEYWORD:(<PUBLIC>|<CLASS>|<EXTENDS>|<STATIC>|<VOID>|<MAIN>|<STRI
    NG>|<IF>|<ELSE>|<DO>|<DOUBLE>|<WHILE>|<NEW>|<LENGTH>|<PRINT>|<INT
    >|<BOOLEAN>|<TRUE>|<FALSE>|<THIS>|<RETURN>)>
}


/* SEPARATORS  */
TOKEN : {
  <LBRACE: "{"> | <RBRACE: "}"> | <LBRACK:"["> | <RBRACK:"]"> | <LPAREN:"(">
  | <RPAREN:")"> | <COMMA:","> | <SEMI:";"> | <DOT:".">
  |<SIGN:(<LBRACE>|<RBRACE>|<LBRACK>|<RBRACK>|<LPAREN>|<RPAREN>|<COMM
  A>|<SEMI>|<DOT>)>
}

/* LITERALS */
TOKEN :
{
     <INTLITERAL:(["0" - "9"])+>|
    <REALLITERAL:((["0"-"9"])+"."(["0"-"9"])*)|((["0"-"9"])*"."(["0"-
    "9"])+)>
}


/*IDENTIFIERS*/
TOKEN :
{
    <ID: (["a"-"z","A"-"Z"])(["a"-"z","A"-"Z","0"-"9","_"])*>
}

/* COMMENTS */
SPECIAL_TOKEN:{
    <#ONE_LINE_COMMENT: "//"(~["\n", "\r"])*("\n"|"\r" |"\r\n")>
    |<#LINES_COMMENT: "/*"((~["*"])|("*"(~["/"])))*"*/">
    |<#FORMAL_COMMENT: "/**"((~["*"])|("*"(~["/"])))*"*/">
    |<COMMENT:(<ONE_LINE_COMMENT>|<LINES_COMMENT>|<FORMAL_COMMENT>)>

}
```

Appendix B. Language Grammars

```
Program → MainClass ClassDecl*
 MainClass → class id { public static void main ( String [] id )
               { Statement }}
 ClassDecl → class id { VarDecl* MethodDecl* }
           → class id extends id { VarDecl* MethodDecl* }
   VarDecl → Type id ;
MethodDecl → public Type id ( FormalList )
               { VarDecl* Statement* return Exp ;}
FormalList → Type id FormalRest*
           →
FormalRest →, Type id
      Type → int []
           → boolean
           → int
           → id
 Statement → { Statement* }
           → if ( Exp ) Statement else Statement
           → while ( Exp ) Statement
           → System.out.println ( Exp ) ;
           → id = Exp ;
           → id [ Exp ]= Exp ;
       Exp → Exp op Exp
           → Exp [ Exp ]
           → Exp . length
           → Exp . id ( ExpList )
           → INTEGER LITERAL
           → true
           → false
           → id
           → this
           → new int [ Exp ]
           → new id ()
           → ! Exp
           → ( Exp )
   ExpList → Exp ExpRest*
           →

  ExpRest  →  ,Exp
```

Appendix C. Exp Structure



Appendix D. jjTree Grammars

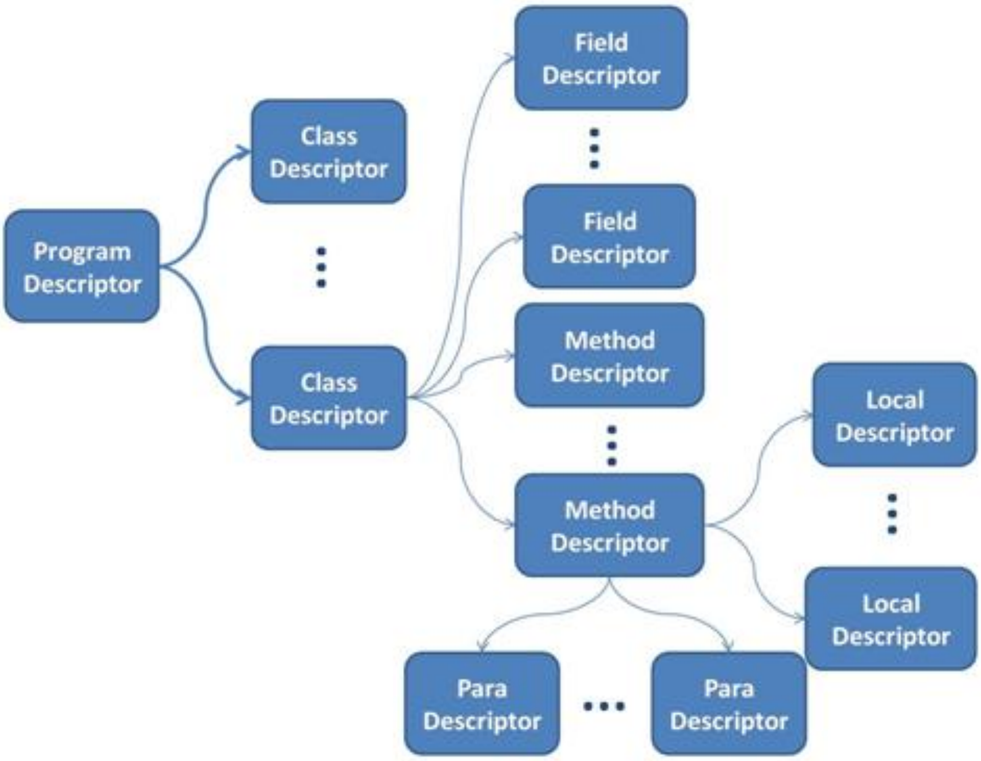
```
void Program() : {}
{
(MainClass()(ClassDecl())*) #Program
}

void MainClass()#MainClass : { }
{
    <CLASS>
    ID()
    <LBRACE>
    <PUBLIC><STATIC><VOID><MAIN>
    <LPAREN>
    <STRING><LBRACK><RBRACK>ID()
    <RPAREN>
    <LBRACE>
    (Statement())*
    <RBRACE>
    <RBRACE>
}

void ClassDecl()#ClassDecl : { }
{
    <CLASS>
    ID()
    (<EXTENDS>ID() #ExtendNode)?
    <LBRACE>
    (Body())*
    <RBRACE>
}

void Body():{}
{
    VarDecl()
    |MethodDecl()
}


void Statement()#StatementNode : {}
{
    IfStatement()
    |WhileStatement()
    |DoWhileStatement()
    |StatementBlock()
    |SingleStatement()
}

void IfStatement()#IfStatementNode : {}
{
    <IF><LPAREN>JudgeExp()<RPAREN>Statement()
    <ELSE>#ElseNodeStatement()
}

void WhileStatement()#WhileStatementNode : {}
{
    <WHILE><LPAREN>JudgeExp()<RPAREN>Statement()
}

void DoWhileStatement()#DoWhileStatement : {}
{
    <DO>Statement()<WHILE><LPAREN>JudgeExp()<RPAREN><SEMI>
}

void StatementBlock() : {}
{
    <LBRACE> (Statement())* <RBRACE>
}

void SingleStatement() #SingleStatementNode: {}
{
    PrintStatement()|AssignStatement()
}

void PrintStatement() #PrintStatementNode : {}
{
    <PRINT><LPAREN>Exp()<RPAREN><SEMI>
}


void VarDecl() #VarDeclNode: {}
{
    Type() ID()<SEMI>
}


void MethodDecl()#MethodNode : {}
```

```
{
    <PUBLIC>
    Type() ID()
    <LPAREN>FormalList()<RPAREN>
    <LBRACE>
    (MethodBody())*
    ReturnStatement()
    <RBRACE>
}

void MethodBody() : {}
{
    LOOKAHEAD(VarDecl())VarDecl()
    |Statement()
}

void FormalList()#FormalListNode : {Token t;String type;}
{
    (type = Type()
    t= ID()
    (FormalRest())*
    {jjtThis.setText(type + " " + t.image);})?
}

void FormalRest()#FormalRestNode : { }
{
    <COMMA>
    Type()
    ID()
}

void ReturnStatement()#ReturnStatementNode : {}
{
    <RETURN>Exp()<SEMI>
}

String Type(): {String type;}
{
    LOOKAHEAD(ObjectType())
    type = ObjectType(){return type;}
    |type = PrimitiveType(){return type;}
}

String PrimitiveType() #PrimitiveTypeNode:{ }
```

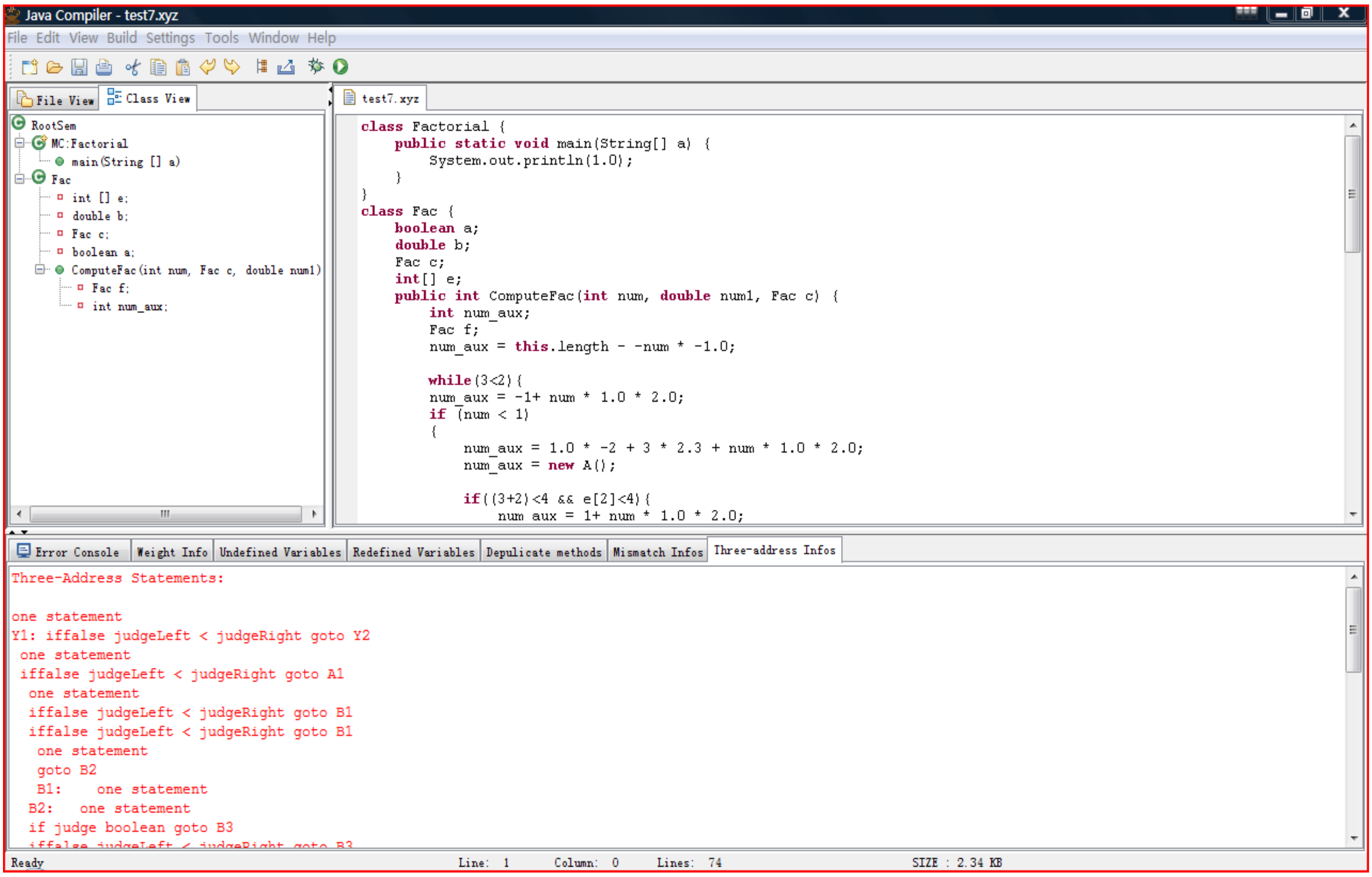
```
{
    t = <INT>
    |t = <DOUBLE>
    |t = <BOOLEAN>
}

String ObjectType() #ObjectTypeNode : { }
{
    <INT><LBRACK><RBRACK>
    |<DOUBLE><LBRACK><RBRACK>
    | ID()
}

//-------------------------------------JudgeExp----------------------
-----------
void JudgeExp() #JudgeExpNode:{}{
    (LOOKAHEAD(LessExp())LessExp()
    |NotExp())
    (<AND>
    (LOOKAHEAD(LessExp())LessExp()
    |NotExp()))*
}

void NotExp() #NotExpNode:{ }{
    (<NOT>)?
    (<ID>
    |<TRUE>
    |<FALSE>)
}

void LessExp() #LessExpNode:{}{
    (ConstantElement()|
    LOOKAHEAD(ArrayElement())ArrayElement()
    |IDElement()
    |ChildExp())
    <LT>
    (ConstantElement()|
    LOOKAHEAD(ArrayElement())ArrayElement()
    |IDElement()
    |ChildExp())
}
//------------------------------Assign-------------------------------

void AssignStatement() #AssignStatementNode : {}
```

```
{
    AssignLeft() <ASSIGN> AssignRight() <SEMI>
}

void AssignLeft() #AssignLeftNode:{}{
    LOOKAHEAD(ArrayElement())ArrayElement()
    |IDElement()
}

void AssignRight()#AssignRightNode:{}{
    LOOKAHEAD(NewExp())
    NewExp()|Exp()
}

//---------------------------------------Exp--------------------------

void Exp() #ExpNode: {}
{
    MultiExp()((<PLUS>MultiExp()#AddExpNode)|(<MINUS>MultiExp()#Minus
ExpNode))*
}

void MultiExp()  : {}
{
    SignExp()(<MUL>SignExp()#MultiExpNode)*
}


void SignExp() : {}
{
    (<MINUS>#SignExpNode)?Element()
}
//----------------------------Elemet----------------------------------
void Element() : { }
{
    ConstantElement()
    |LOOKAHEAD(ArrayElement())ArrayElement()
    |LOOKAHEAD(CallExp())CallExp()
    |IDElement()
    |ChildExp()
}

void CallExp() #CallExpNode : {}
{
    (This()|ID()|NewExp())<DOT>((ID()<LPAREN>ExpList()<RPAREN>)|<LENGTH>)
}

void ChildExp() #ChildExpNode : {}
{
    <LPAREN>Exp()<RPAREN>
}

void ArrayElement() #ArrayElementNode : { }
{
    <ID> <LBRACK>Exp()<RBRACK>
}

void IDElement() #IDElementNode : { }
{
    <ID>
}

void ConstantElement() #ConstantElementNode :{ }{
    <INTLITERAL>
    |<REALLITERAL>
}
//-------------------------------------------------------------------------------

void NewExp() #NewExpNode : { }
{
    <NEW>
    (IntArray()
    |DoubleArray()
    |ID()<LPAREN><RPAREN>)
}

void IntArray() : {}{
    <INT><LBRACK>Exp()<RBRACK>
}

void DoubleArray() :{}{
    <DOUBLE><LBRACK>Exp()<RBRACK>
}

void IDExp() #IDExpNode: {}
{
    Object()
    ( <LBRACK>Exp()<RBRACK>  #IndexNode
       |LOOKAHEAD(2)<DOT><LENGTH>             #DotLengthNode
       |<DOT>ID()<LPAREN>ExpList()<RPAREN>    #InvokeFunctionNode)?
}

void Object()  : {}
{
    This()
    |ID()
    |NewExp()
}

Token ID() #IDNode:{ }{
    <ID>
}

void This() #ThisNode:{}{
    <THIS>
}

void ExpList() #ExpListNode: {}
{
    (Exp()(<COMMA>Exp())*)?

}
```

Appendix E. Symbol Table



Appendix F. Snapshot



```
RootSem
  MC:Factorial
    main(String [] a)
  Fac
    int [] e;
    double b;
    Fac c;
    boolean a;
    ComputeFac(int num, Fac c, double num1)
      Fac f;
      int num_aux;
```

```
class Factorial {
    public static void main(String[] a) {
        System.out.println(1.0);
    }
}
class Fac {
    boolean a;
    double b;
    Fac c;
    int[] e;
    public int ComputeFac(int num, double num1, Fac c) {
        int num_aux;
        Fac f;
        num_aux = this.length - -num * -1.0;

        while(3<2){
        num_aux = -1+ num * 1.0 * 2.0;
        if (num < 1)
        {
            num_aux = 1.0 * -2 + 3 * 2.3 + num * 1.0 * 2.0;
            num_aux = new A();

            if((3+2)<4 && e[2]<4){
                num aux = 1+ num * 1.0 * 2.0;
```

```
Three-Address Statements:

one statement
Y1: iffalse judgeLeft < judgeRight goto Y2
 one statement
 iffalse judgeLeft < judgeRight goto A1
  one statement
  iffalse judgeLeft < judgeRight goto B1
  iffalse judgeLeft < judgeRight goto B1
   one statement
   goto B2
   B1:    one statement
  B2:   one statement
  if judge boolean goto B3
  iffalse judgeLeft < judgeRight goto B3
```

```
Ready                Line:  1      Column:  0      Lines:  74                SIZE : 2.34 KB
```