PLD Final

Chris Fenton (fenwil28)

Initialization Declarations

Example Program

```
int main ( ) {
    int n = 3;
    int i = 1;
    int f = 1;
    while (i < n) {
        i = i + 1;
        f = f * i;
    }
}
```

Lexer Changes

No new tokens needed

BNF Changes

Declaration -> BaseDeclaration | AssignmentDeclaration

BaseDeclaration -> Type Identifier;

AssignmentDeclaration -> Type Identifier Assignment;

AST Changes

Declaration = BaseDeclaration | AssignmentDeclaration

BaseDeclaration = Variable v; Type t

AssignmentDeclaration = Variable v; Type t; Assignment a

Implementation

Parser.java

```java
private void declaration (Declarations ds) {
  // Declaration  --> Type Identifier { , Identifier } ;
  // student exercise
  Type type;
  String id;

  type = type();

  while (!token.type().equals(TokenType.Semicolon)) {
    id = match(TokenType.Identifier);

    if (token.type().equals(TokenType.Assign)) {
      ds.add(new AssignmentDeclaration(new Variable(id),
type, assignment()));
    } else {
      ds.add(new BaseDeclaration(new Variable(id), type))
;
    }

  }
  match(TokenType.Semicolon);
}
```

AbstractSyntax.java

```
abstract class Declaration {
    // Declaration = BaseDeclaration | AssignmentDeclarat
ion
}

class BaseDeclaration extends Declaration{
// Declaration = Variable v; Type t
    Variable v;
    Type t;

    BaseDeclaration (Variable var, Type type) {
        v = var; t = type;
    } // declaration */

}

class AssignmentDeclaration extends Declaration {
  Variable v;
  Type t;
  Assignment a;

  AssignmentDeclaration (Variable var, Type type, Assignm
ent ass) {
    v = var; t = type; a = ass;
  }
}
```

Tuples

Example Program

```
int main ( ) {
    tuple n = <<3,4>>;
}
```

Lexer Changes

New token types: leftTupleTok and rightTupleTok

Token.java

```
public static final Token leftTupleTok = new Token(TokenT
ype.LeftAngle, "<<");
public static final Token rightTupleTok = new Token(Token
Type.RightAngle, ">>");
```

Lexer.java

changes to next()

```
case '<':
  return chkTuple('<', Token.leftTupleTok,
      Token.ltTok,
      Token.lteqTok);
case '>':
  return chkTuple('>', Token.rightTupleTok,
      Token.gtTok,
      Token.gteqTok);
```

chkTuple()

```
private Token chkTuple(char c, Token one, Token two, Toke
n three) {
    ch = nextChar();
    if (ch == c) {
        return one;
    }
    ch = nextChar();
    if (ch != '=') {
      return two;
    }
    return three;
}
```

BNF

Primary -> Identifier [[Expression]] | Literal | (Expression) | Type (Expression)

| <<Literal,Literal>>

Tuple -> <<Value,Value>>

AST

Tuple = Value f; Value s;

Implementation

AbstractSyntax.java

```java
class Tuple extends Expression {
  Value f;
  Value s;

  Tuple(Value first, Value second) {
    f = first; s = second;
  }
}
```

Parser.java

```
private Expression primary () {
    Expression e = null;
    if (token.type().equals(TokenType.Identifier)) {
      e = new Variable(match(TokenType.Identifier));
    } else if (isLiteral()) {
      e = literal();
    } else if (token.type().equals(TokenType.LeftParen))
{
      token = lexer.next();
      e = expression();
      match(TokenType.RightParen);
    } else if (isType( )) {
      Operator op = new Operator(match(token.type()));
      match(TokenType.LeftParen);
      Expression term = expression();
      match(TokenType.RightParen);
      e = new Unary(op, term);
    // ***** Tuple Code here *****
    } else if (token.type().equals(TokenType.LeftAngle))
{
      token = lexer.next();
      Value f = literal();
      match(TokenType.Comma);
      Value s = literal();
      match(TokenType.RightAngle);
      e = new Tuple(f, s);
    // ***** End Tuple Code *****
    } else error("Identifier | Literal | ( | Type");
    return e;
  }
```