Advanced Object-Oriented Programming, Spring 2016

Homework Assignment #4

Due midnight Wednesday, May 25, 2016

Instructions

1. If any question is unclear, please ask for a clarification.
2. You may try to reuse as much of the source code supplemented as possible.
3. Unless stated otherwise, all the line numbers for the program listings are for reference only.
4. You are required to do all the homework assignments on Linux and use g++ version 4.9.2 or later.
5. You are required to give your TA a demo of your program. Make sure that your program can compile and run on the server machine, which will be used for the demo.
6. For the program that you write, you are required to include a Makefile. Otherwise, your homework will not be graded—meaning that you will receive zero marks.
7. Unless stated otherwise, you are required to work on the homework assignment individually.
8. No late homework will be accepted.

Programming Project

This assignment requires that you implement a scanner using inheritance in C++ for the language defined in Appendix A of the dragon book, second edition,[1] the source and class hierarchy of which are given in Listing 1 to Listing 8.

Listing 1: Main.java

```java
package main;

import java.io.*;
import lexer.*;

public class Main {
    public static void main(String[] args) throws IOException
    {
```

[1]Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman, *Compilers: Principles, Techniques, and Tools, Second Edition*, Addison-Wesley, 2007. Fortunately, you don't have to understand the material as far as the homework assignment is concerned. Instead, all you have to do is figure out how the Java version works and follow the class hierarchy for the C++ implementation.

```java
        Lexer lexer = new Lexer();
        try {
            while (true) {
                Token t = lexer.scan();
                System.out.println(t.toString());
            }
        }
        catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Listing 2: Lexer.java

```java
package lexer;

import java.io.*;
import java.util.*;

public class Lexer {
    public static int line = 1;
    char peek = ' ';
    Hashtable words = new Hashtable();
    void reserve(Word w)
    {
        words.put(w.lexeme, w);
    }
    public Lexer()
    {
        reserve(new Word("if",          Tag.IF)         );
        reserve(new Word("else",        Tag.ELSE)       );
        reserve(new Word("while",       Tag.WHILE)      );
        reserve(new Word("do",          Tag.DO)         );
        reserve(new Word("break",       Tag.BREAK)      );
        reserve(Word.True);
        reserve(Word.False);
        reserve(Type.Int);
        reserve(Type.Char);
        reserve(Type.Bool);
        reserve(Type.Float);
    }
    void readch() throws IOException
    {
        int i = System.in.read();
        if (i != -1)
            peek = (char) i;
        else
            throw new IOException("End of file reached");
    }
    boolean readch(char c) throws IOException
    {
        readch();
        if (peek != c)
            return false;
        peek = ' ';
        return true;
    }
    public Token scan() throws IOException
    {
        for ( ; ; readch()) {
            if (peek == ' ' || peek == '\t')
                continue;
            else if (peek == '\n')
```

```
                line = line + 1;
            else
                break;
        }
        switch (peek) {
        case '&':
            if (readch('&'))
                return Word.and;
            else
                return new Token('&');
        case '|':
            if (readch('|'))
                return Word.or;
            else
                return new Token('|');
        case '=':
            if (readch('='))
                return Word.eq;
            else
                return new Token('=');
        case '!':
            if (readch('='))
                return Word.ne;
            else
                return new Token('!');
        case '<':
            if (readch('='))
                return Word.le;
            else
                return new Token('<');
        case '>':
            if (readch('='))
                return Word.ge;
            else
                return new Token('>');
        }

        if (Character.isDigit(peek)) {
            int v = 0;
            do {
                v = 10 * v + Character.digit(peek, 10);
                readch();
            } while (Character.isDigit(peek));
            if (peek != '.')
                return new Num(v);
            float x = v;
            float d = 10;
            for (;;) {
                readch();
                if (!Character.isDigit(peek))
                    break;
                x = x + Character.digit(peek, 10) / d;
                d = d * 10;
            }
            return new Real(x);
        }

        if (Character.isLetter(peek)){
            StringBuffer b = new StringBuffer();
            do {
                b.append(peek);
                readch();
            } while (Character.isLetterOrDigit(peek));
            String s = b.toString();
            Word w = (Word) words.get(s);
            if (w != null)
                return w;
            w = new Word(s, Tag.ID);
            words.put(s, w);
            return w;
        }

        Token tok = new Token(peek);
        peek = ' ';
        return tok;
    }
}
```

Listing 3: Tag.java

```
package lexer;

public class Tag {
    public final static int
        AND     = 256,
        BASIC   = 257,
        BREAK   = 258,
        DO      = 259,
        ELSE    = 260,
        EQ      = 261,
        FALSE   = 262,
        GE      = 263,
        ID      = 264,
        IF      = 265,
        INDEX   = 266,
        LE      = 267,
        MINUS   = 268,
        NE      = 269,
        NUM     = 270,
        OR      = 271,
        REAL    = 272,
        TEMP    = 273,
        TRUE    = 274,
        WHILE   = 275;
}
```

Listing 4: Token.java

```
package lexer;

public class Token {
    public final int tag;
    public Token(int t)
    {
        tag = t;
    }
    public String toString()
    {
        return "" + (char) tag;
    }
}
```

Listing 5: Word.java

```
package lexer;

public class Word extends Token {
```

```java
    public String lexeme = "";
    public Word(String s, int tag)
    {
        super(tag);
        lexeme = s;
    }
    public String toString()
    {
        return lexeme;
    }
    public static final Word
        and     = new Word("&&",        Tag.AND),
        or      = new Word("||",        Tag.OR),
        eq      = new Word("==",        Tag.EQ),
        ne      = new Word("!=",        Tag.NE),
        le      = new Word("<=",        Tag.LE),
        ge      = new Word(">=",        Tag.GE),
        minus   = new Word("minus",     Tag.MINUS),
        True    = new Word("true",      Tag.TRUE),
        False   = new Word("false",     Tag.FALSE),
        temp    = new Word("t",         Tag.TEMP);
}
```

Listing 6: Type.java

```java
package lexer;

public class Type extends Word {
    public int width = 0;
    public Type(String s, int tag, int w)
    {
        super(s, tag);
        width = w;
    }
    public static final Type
        Int     = new Type("int",       Tag.BASIC, 4),
        Float   = new Type("float",     Tag.BASIC, 8),
        Char    = new Type("char",      Tag.BASIC, 1),
        Bool    = new Type("bool",      Tag.BASIC, 1);
    public static boolean numeric(Type p)
    {
        if (p == Type.Char || p == Type.Int || p == Type.Float)
            return true;
        else
            return false;
    }
    public static Type max(Type p1, Type p2)
    {
        if (!numeric(p1) || !numeric(p2))
            return null;
        else if (p1 == Type.Float || p2 == Type.Float)
            return Type.Float;
        else if (p1 == Type.Int || p2 == Type.Int)
            return Type.Int;
        else
            return Type.Char;
    }
}
```

Listing 7: Num.java

```java
package lexer;

```

```java
public class Num extends Token {
    public final int value;
    public Num(int v)
    {
        super(Tag.NUM);
        value = v;
    }
    public String toString()
    {
        return "" + value;
    }
}
```

Listing 8: Real.java

```java
package lexer;

public class Real extends Token {
    public final float value;
    public Real(float v)
    {
        super(Tag.REAL);
        value = v;
    }
    public String toString()
    {
        return "" + value;
    }
}
```

To make it easier for you to test your implementation, also given are the test programs (Listings 9 and 11) and the output of the scanner in C++ (Listings 10 and 12). Note that the output of the scanner in C++ is different from that in Java.

Listing 9: Test program 1

```
{
    int i;
    int j;
    float v;
    float x;
    float[100] a;
    while (true) {
        do i = i+1; while (a[i] < v);
        do j = j-1; while (a[j] > v);
        if (i >= j) break;
        x = a[i];
        a[i] = a[j];
        a[j] = x;
    }
}
```

Listing 10: Output of the test program 1

```
Token: {        ({)
Token: int      (BASIC)
Token: i        (ID)
Token: ;        (;)
Token: int      (BASIC)
Token: j        (ID)
Token: ;        (;)
Token: float    (BASIC)
```

```
Token: v        (ID)
Token: ;        (;)
Token: float    (BASIC)
Token: x        (ID)
Token: ;        (;)
Token: float    (BASIC)
Token: [        ([)
Token: 100      (NUM)
Token: ]        (])
Token: a        (ID)
Token: ;        (;)
Token: while    (WHILE)
Token: (        (()
Token: true     (TRUE)
Token: )        ())
Token: {        ({)
Token: do       (DO)
Token: i        (ID)
Token: =        (=)
Token: i        (ID)
Token: +        (+)
Token: 1        (NUM)
Token: ;        (;)
Token: while    (WHILE)
Token: (        (()
Token: a        (ID)
Token: [        ([)
Token: i        (ID)
Token: ]        (])
Token: <        (<)
Token: v        (ID)
Token: )        ())
Token: ;        (;)
Token: do       (DO)
Token: j        (ID)
Token: =        (=)
Token: j        (ID)
Token: -        (-)
Token: 1        (NUM)
Token: ;        (;)
Token: while    (WHILE)
Token: (        (()
Token: a        (ID)
Token: [        ([)
Token: j        (ID)
Token: ]        (])
Token: >        (>)
Token: v        (ID)
Token: )        ())
Token: ;        (;)
Token: if       (IF)
Token: (        (()
Token: i        (ID)
Token: >=       (GE)
Token: j        (ID)
Token: )        ())
Token: break    (BREAK)
Token: ;        (;)
Token: x        (ID)
Token: =        (=)
Token: a        (ID)
Token: [        ([)
Token: i        (ID)
Token: ]        (])
Token: ;        (;)
Token: a        (ID)
Token: [        ([)
Token: i        (ID)
Token: ]        (])
Token: =        (=)
Token: a        (ID)
Token: [        ([)
Token: j        (ID)
Token: ]        (])
Token: ;        (;)
Token: a        (ID)
Token: [        ([)
Token: j        (ID)
Token: ]        (])
Token: =        (=)
Token: x        (ID)
Token: ;        (;)
Token: }        (})
Token: }        (})
End of file reached
```

Listing 11: Test program 2

```
{
    int i;
    int j;
    float[10][10] a;
    i = 0;
    while ( i < 10 ) {
        j = 0;
        while ( j < 10 ) {
            a[i][j] = 0;
            j = j+1;
        }
        i = i+1;
    }
    i = 0;
    while ( i < 10 ) {
        a[i][j] = 1;
        i = i+1;
    }
}
```

Listing 12: Output of the test program 2

```
Token: {        ({)
Token: int      (BASIC)
Token: i        (ID)
Token: ;        (;)
Token: int      (BASIC)
Token: j        (ID)
Token: ;        (;)
Token: float    (BASIC)
Token: [        ([)
Token: 10       (NUM)
Token: ]        (])
Token: [        ([)
Token: 10       (NUM)
Token: ]        (])
Token: a        (ID)
Token: ;        (;)
Token: i        (ID)
```

```
Token: =        (=)
Token: 0        (NUM)
Token: ;        (;)
Token: while    (WHILE)
Token: (        (()
Token: i        (ID)
Token: <        (<)
Token: 10       (NUM)
Token: )        ())
Token: {        ({)
Token: j        (ID)
Token: =        (=)
Token: 0        (NUM)
Token: ;        (;)
Token: while    (WHILE)
Token: (        (()
Token: j        (ID)
Token: <        (<)
Token: 10       (NUM)
Token: )        ())
Token: {        ({)
Token: a        (ID)
Token: [        ([)
Token: i        (ID)
Token: ]        (])
Token: [        ([)
Token: j        (ID)
Token: ]        (])
Token: =        (=)
Token: 0        (NUM)
Token: ;        (;)
Token: j        (ID)
Token: =        (=)
Token: j        (ID)
Token: +        (+)
Token: 1        (NUM)
Token: ;        (;)
Token: }        (})
Token: i        (ID)
Token: =        (=)
Token: i        (ID)
Token: +        (+)
Token: 1        (NUM)
Token: ;        (;)
Token: }        (})
Token: i        (ID)
Token: =        (=)
Token: 0        (NUM)
Token: ;        (;)
Token: while    (WHILE)
Token: (        (()
Token: i        (ID)
Token: <        (<)
Token: 10       (NUM)
Token: )        ())
Token: {        ({)
Token: a        (ID)
Token: [        ([)
Token: i        (ID)
Token: ]        (])
Token: [        ([)
Token: j        (ID)
Token: ]        (])
Token: =        (=)
Token: 1        (NUM)
Token: ;        (;)
Token: i        (ID)
Token: =        (=)
Token: i        (ID)
Token: +        (+)
Token: 1        (NUM)
Token: ;        (;)
Token: }        (})
Token: }        (})
End of file reached
```

Grading Policy

The grading policy for this assignment is as follows:

- Make sure that a **Makefile**, which contains at least three targets—**all**, **dep**, and **clean**—is provided. Otherwise, the grade for your program will be zero.
- 100 points if your program compiles without errors and warnings and runs correctly.