词法分析实验

姓　　名：高　　月

学　　号：1120141944

班　　级：07121402

指导老师：计 卫 星

目录

一、实验目的

通过实践环节深入理解与编译实现有关的形式语言理论基本概念，掌握编译程序构造的一般原理、基本设计方法和主要实现技术，并通过运用自动机理论解决实际问题，从问题定义、分析、建立数学模型和编码的整个实践活动中逐步提高软件设计开发的能力。

二、实验内容

该实验以 C 语言作为源语言，构建 C 语言的词法分析器，对于给定的测试程序，输出 XML 格式的属性字符流。

三、实验环境

本次实验系统环境等信息如下表所示

OS	IDE	Java SDK
macOS Sierra 10.12.4	IntelliJ IDEA 17.1	1.8 update 101

四、实验步骤

4.1 熟悉 BIT-MiniCC 框架

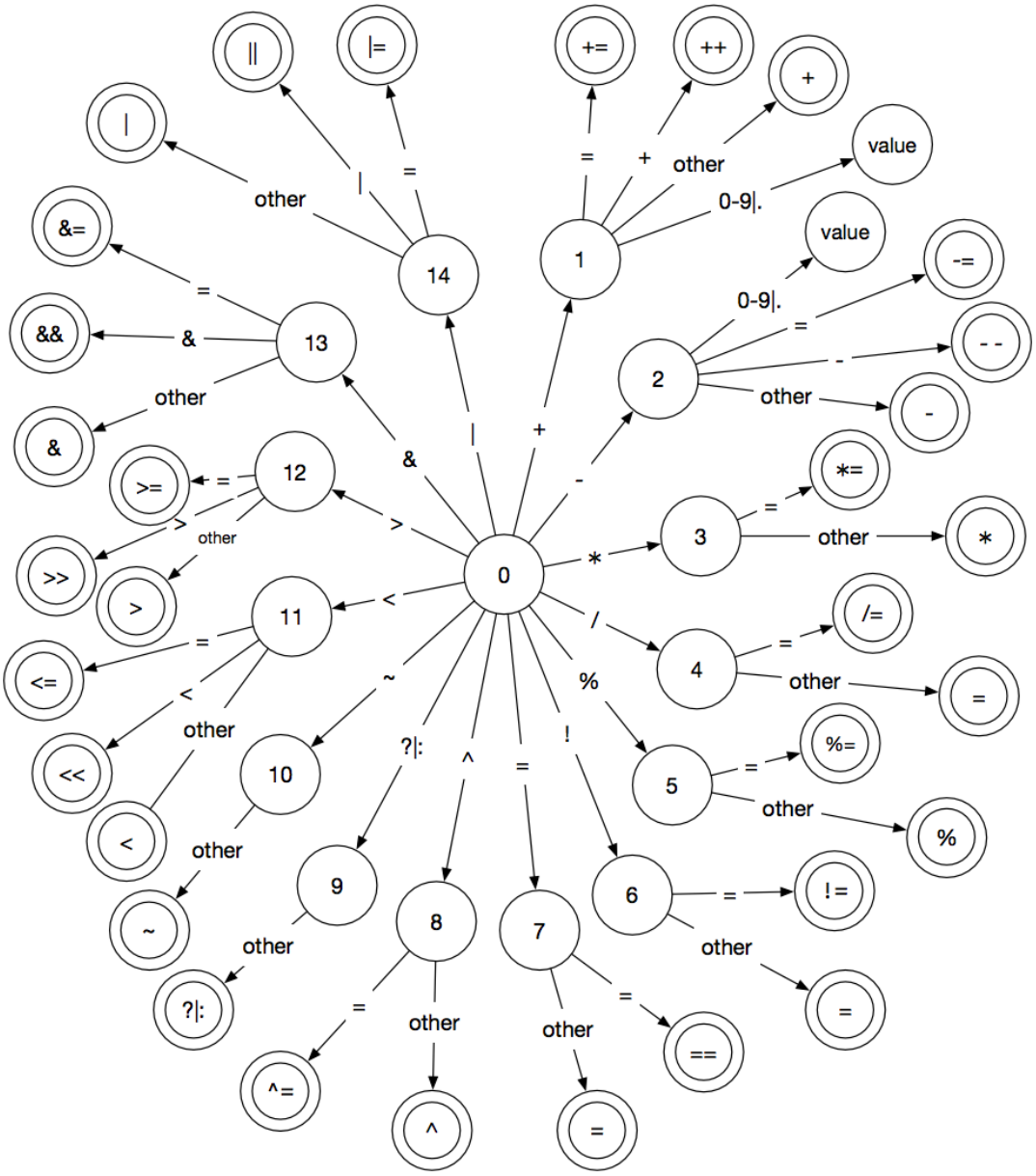
从 GitHub 克隆实验框架仓库，运行实验框架，并查看框架自带词法分析输出属性字符流格式。配置 config.xml 文件，如下图所示：

```
<phase skip="true" type="java" path="" name="pp" />
<phase skip="false" type="java" path="bit.minisys.minicc.GScanner" name="scanning" />
```

4.2 分析 C 语言词法

4.2.1 运算符分析

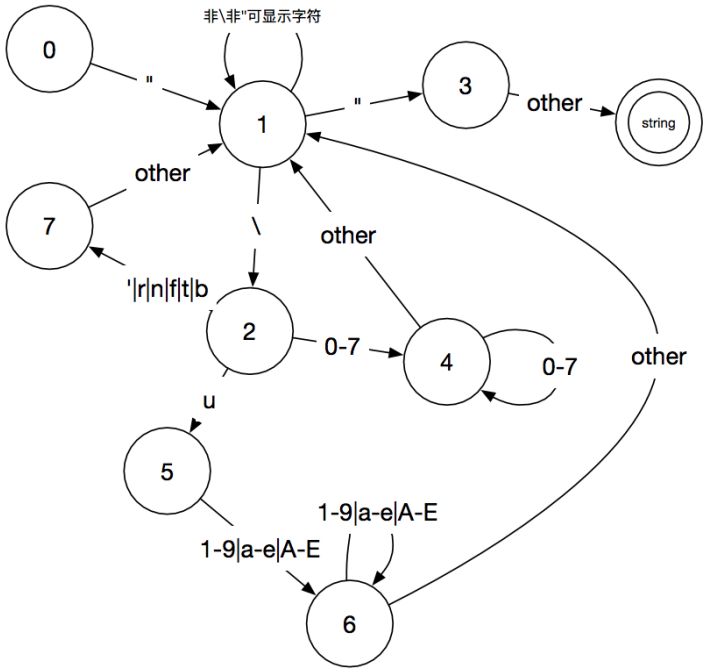
状态转换图如下图所示：



注：其中 other 为其他任意字符，后同。

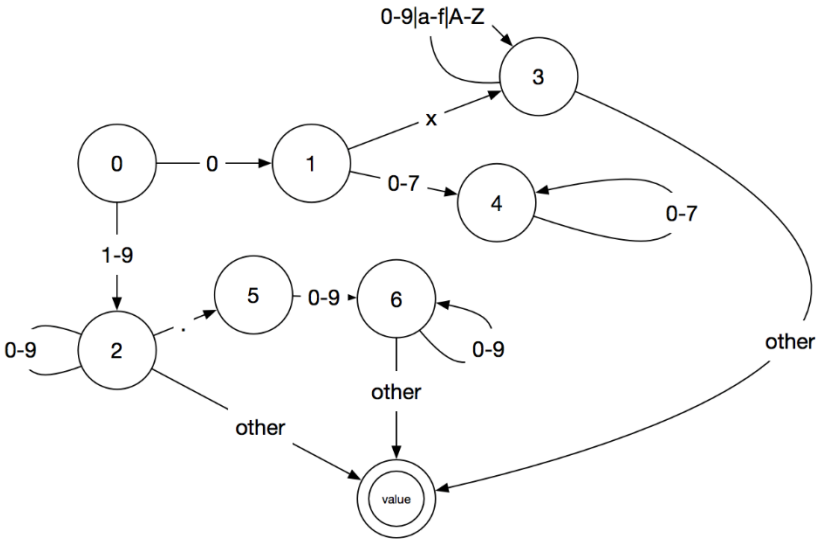
4.2.2 字符串分析

状态转化图如下图所示



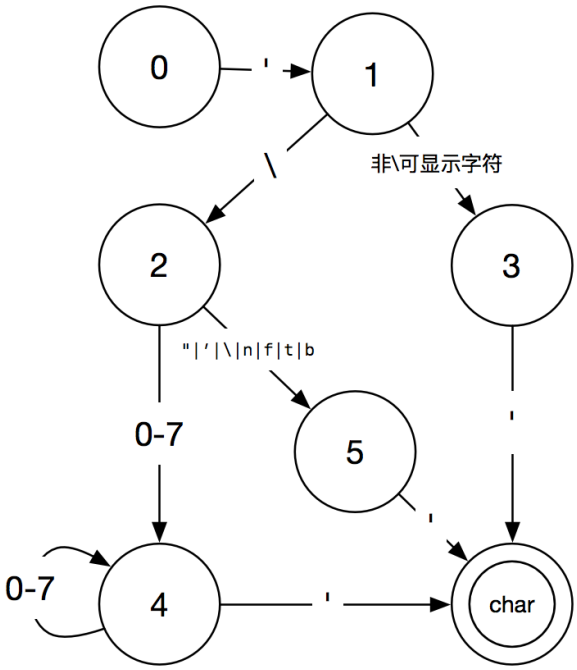
4.2.3 数值分析

状态转化图如下图所示：



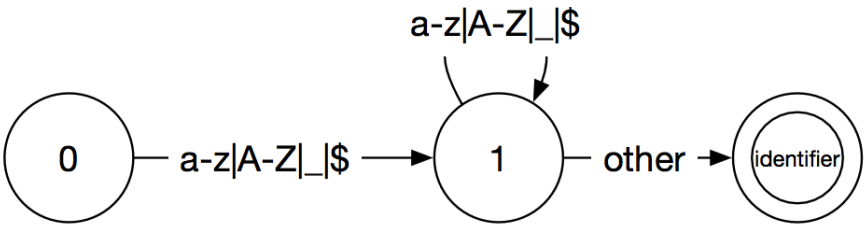
4.2.4 字符分析

状态转化图如下图所示：



4.2.5 标识符分析

状态转换表如下图所示：



4.2.6 分隔符分析

分隔符判断比较容易，状态转换图从略。

4.2.7 空白符分析

对待空白符，直接忽略即可，而空白符前后根据各自类型分别处理即可。

4.3 程序实现

4.3.1 整体设计

本次实验中创建了三个类：

`GTokenInfo` 类：负责存储一个 token 的相关信息；

`GLexicalAnalyzer` 类：负责词法分析；

`GScanner` 类 :负责整个 Scanner 的框架 ,包含文件的读写 ,词法分析器的调用。

4.3.2 `GTokenInfo` 类

`GTokenInfo` 类用于存储 token 的信息，主要实现如下：

```
public class GTokenInfo {
    // token type
    public static int NULL_TYPE = (0x0000);
    public static int OPERATOR_TYPE = (0x0001 << 0);
    public static int STRING_TYPE = (0x0001 << 1);
    public static int VALUE_TYPE = (0x0001 << 2);
    public static int CHAR_TYPE = (0x0001 << 3);
    public static int IDENTIFIER_TYPE = (0x0001 << 4);
    public static int KEYWORD_TYPE = (0x0001 << 5);
    public static int SEPARATOR_TYPE = (0x0001 << 6);
    public static int ERROR_TYPE = -1;

    public int number;          // token's number
    public int line;            // token's line
    public String value;        // token's value
    public int type;            // token's type
    public boolean isValid;     // if token is valid
}
```

4.3.3 GLexicalAnalyzer 类

GLexicalAnalyzer 类，主要用于词法分析，每个 GLexicalAnalyzer 类实例只处理一行，包含各词法分析方法，主要实现如下：

```
public class GLexicalAnalyzer {
    private String currentProcessingLine;
    private int lineIndex = 0;
    private int state = 0;

    GLexicalAnalyzer(String sCurrentLine) { currentProcessingLine = sCurrentLine; }

    char getNextChar() {...}

    char backToLastChar() {...}

    boolean processLine() {...}

    GTokenInfo operatorScanner(char ch) {...}

    GTokenInfo stringScanner(char ch) {...}

    GTokenInfo valueScanner(char ch) {...}

    GTokenInfo charScanner(char ch) {...}

    GTokenInfo identifierScanner(char ch) {...}

    GTokenInfo separatorScanner(char ch) {...}

    void blankScanner(char ch) {...}

    boolean specificSymbolScanner(char ch) {...}
}
```

注：由于篇幅限制，上图只截取了类的框架，完整的源代码在附件中，后同。

GLexicalAnalyzer 的 processLine() 方法是词法分析的入口，其根据状态调用各类别词法分析函数。

4.3.4 GScanner 类

GScanner 类实现了 IMiniCCScanner 接口，用于处理输入文件、输出 XML 文件、保存 token、报错、完成创建 GLexicalAnalyzer 类实例的创建及实例的 processLine() 方法的调用等，主要实现如下：

```
public class GScanner implements IMiniCCScanner {

    private static int currentLineNum = 0;
    private static int currentTokenNum = 0;
    private static List<GTokenInfo> tokenLab;

    private String sCurrentLine;
    private GLexicalAnalyzer lexicalAnalyzer;

    public static void addTokenToLab(GTokenInfo tokenInfo) {...}

    public static void printErrorPosition(int errorColumnNum) {...}

    public static String typeIntToString(int typeInt) {...}

    public static void saveToXML(String oXMLFile) {...}

    void parseFile(String iFile) throws Exception {...}

    @Override
    public void run(String iFile, String oFile) throws Exception {
        tokenLab = new ArrayList<>();

        parseFile(iFile);
        saveToXML(oFile);
    }
}
```

五、实验结果

5.1 IDE 的配置

为便于运行测试，在 IntelliJ IDEA 中配置主函数参数即输出文件，主菜单 Run --> Edit Configurations，配置如下：

5.2 测试所用 C 语言代码

测试的 C 语言代码如下图所示：

```c
int main( ) {
    int a = 10;
    int x = 0x10;
    float b = 20.0;
    int c = 10e5;
    char * str = "123";
    char c = 'c';
    return a + b ;
}
```

5.2 测试输出的 XML 文件

测试输出的属性字符流如下图所示：

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project name="test">
  <tokens>
    <token>
      <number>1</number>
      <line>1</line>
      <value>int</value>
      <type>keyword</type>
      <valid>true</valid>
    </token>
    <token>
      <number>2</number>
      <line>1</line>
      <value>main</value>
      <type>identifier</type>
      <valid>true</valid>
    </token>
    <token> ••• </token>
    <token> ••• </token>
```

注：由于篇幅限制只截取了部分，完整的 XML 文件在附件中。

六、实验心得

通过本次实验深入理解了与编译实现有关的形式语言定义的基本概念，掌握了词法分析程序构造的一般原理、基本设计方法。熟悉了通过自动机理论解决实际问题，提高了软件设计开发能力。

本次实验相对比较复杂，难点在于正确构造 C 语言词法分析的状态转化图，以及代码是实现时，多种负责情况的判定及处理。

通过本次实验对较大的项目的构造有了更深的理解，包括变量命名，模块化编程，代码风格，以及整个项目的组织等实际问题。

总之，通过本次实验，从理论到实践上对自动机理论、编译程序构造理论和方法有了更深层的理解。