编译原理实验报告

——Lexer of PCAT

1 实验目的

- 学习 PCAT 语言和 Lex 的用法；
- 使用 Lex 为 PCAT 编写词法分析器。

2 环境配置

在进行实验之前需要对系统环境进行配置。本组成员中使用的操作系统有 Ubuntu 与 OS X 两种，因此在本部分会对两种情况都进行讨论。实际上对于类 Unix 系统而言，配制方法是极其类似的。

此外，本实验提供的 C++代码是于 1997 年编写的，并不符合现代的 C++语言规范。因此代码页只能在非常早期的 GCC 2.92 版本上编译。在现代的操作系统上安装 GCC 早期版本已经十分困难，因此本小组尝试在原代码的基础上进行修改，得到了一份符合现代 C++标准的代码。修改详见代码。

2.1 编译器

本实验可以采用 G++或是 LLVM 前端 Clang++作为编译器。对于 G++，已测试过在 4.2 及以上的版本中可以通过编译；对于 Clang++，只测试过 LLVM 5.1（Clang 503.0.38），但也应该可以运行在绝大多数版本之上。

对于 Ubuntu，需要使用 Apt 包管理器安装编译器，命令如下：

```
apt-get install gcc g++
```

对于 OS X，可以通过 Xcode 发行包安装编译器，也可以使用 Homebrew 安装 G++或 Clang++。若通过 Xcode 安装编译器，则需要在 Xcode 中下载命令行工具包；若通过 Homebrew 安装编译器，需要运行如下命令之一：

```
brew install gcc49
brew install llvm
```

2.2 词法分析器生成器

实验采用词法分析器生成器 Bison，它是一款兼容 Lex 的生成器。通过输入词法规则文件，就可以生成词法分析器的 C 语言源代码。

对于 Ubuntu，使用 Apt 包管理器安装 Bison，命令如下：

```
apt-get install bison
```

对于 OS X，若安装了 Xcode 命令行工具，则 Bison 已安装好。否则可以使用 Homebrew 安装：

```
brew install bison
```

2.3 其他依赖库

本实验的代码中使用到了 Boehm-Demers-Weiser 垃圾回收器（GC）库。实验材料中包含了 GC 的较早版本。在此采用了最新版本的 GC 库。

对于 Ubuntu，使用 Apt 包管理器安装 GC，命令如下：

```
apt-get install libgc1c2 libgc-dev
```

对于 OS X，使用 Homebrew 安装，命令如下：

```
brew install boemhgc
```

2.4 修改源码与 Makefile 文件

采用已修改后的源码，需要额外对代码进行如下处理：

1. 修改 Makefile 文件：

对于 Ubuntu，设置 GCC=g++；对于 OS X，设置 GCC=g++或 clang++。

编辑 lexer 目标的生成指令，添加 GC 的连接选项：

```
$(GCC) $(CXXFLAGS) main.cc basic.o -lgc -o lexer
```

2. 修改 basic.h 文件：

替换 gc_cpp.h 的引用：

```
#include <gc/gc_cpp.h>
```

添加引用：

```
#include <stdarg.h>
```

修改后的 Makefile 文件如下（适用于 OS X）：

```
GCC = clang++
CXXFLAGS = -g -lgc
LDFLAGS += -v
YACC = bison
LEX = flex
INCLUDE = basic.h basic.templates

lexer: main.cc basic.o pcat.yy.c
   $(GCC) $(CXXFLAGS) main.cc basic.o -lgc -o lexer

basic.o: basic.cc $(INCLUDE)
    $(GCC) $(CXXFLAGS) -c basic.cc

pcat.yy.c: pcat.lex
      $(LEX) pcat.lex
      mv lex.yy.c pcat.yy.c

clean:
   /bin/rm -f *.o *~ pcat.yy.c lexer core
```

2.5 编译运行

至此对环境的配置已经完成。完成 pcat.lex 文件，并在目录下执行 make，即可生成词法分析器并编译。

生成的词法分析器接受一个命令行参数。把实验文件中 tests 目录下的文件输入到词法分析器，即可得到分析后的 token 表。

3 PCAT 语言

PCAT 语言是一个由 pascal 衍生而来的小型语言。在 The PCAT Programming Language Reference Manual 中有详细介绍。下面是有关 PCAT 语法的解释。

- PCAT 的字符集是标准 ANCII 字符集，且对大小写敏感。
- 空白符是用来分割 token 的，多余的空白符是可以被忽略的。两个相邻的 keywords 或者 identifiers 之间，keyword 和 number 之间，以及 identifiers 和 number 之间都要有空白符进行分割。

- 注释是在(*和*)之间，注释本身不能嵌套，注释内可以是任意字符。
- Token 可以是 reserved keywords、constants、identifiers、operators 和 delimiters。
- Reverved keywords 是关键字，必须用大写，下面是关键字列表：

AND、ARRAY、BEGIN、BY、DIV、DO、ELSE、ELSIF、END、EXIT、FOR、IF、IS、LOOP、 MOD、NOT、OF、OR、PROCEDURE、PROGRAM、READ、RECORD、RETURN、THEN、TO、TYPE、VAR、WHILE、WRITE

- Constants 可以是整数、实数或者字符串。整数必须在 0~2147483647 范围内。实数要包含一个小数点，小数点前至少有一个数字。字符串要以双引号括起来，长度不能大于 255，切不能包含制表符和换行。
- Identifiers 是一个由字母和数字组成的字符串，第一位必须是字母，长度不能超过 255，且不能是 keywords。
- Operator 是 : = + - * / < <= > >= = <> 其中的一个。
- Delimiter 是 : ; , . ( ) [ ] { } 其中的一个。

4 LEX 描述

首先简要介绍一下 Lex. 用户只需将想要匹配的模式预先设置好，交给 Lex 直接判断就可以了。

4.1 使用到的 Lex 变量

yytext 字符串内容，用(char*)存储；

yyleng 字符串长度。

4.2 Lex 程序格式

分为三个部分，每个部分使用%%分隔。

a. C 和 Lex 全局声明/定义

b. Lex 匹配规则

c. C 代码

4.3 Lex 的各种符号规则

Character	Meaning
A-Z, 0-9, a-z	Characters and numbers that form part of the pattern.
.	Matches any character except \n.
-	Used to denote range. Example: A-Z implies all characters from A to Z.
[ ]	A character class. Matches *any* character in the brackets. If the first character is ^ then it indicates a negation pattern. Example: [abC] matches either of a, b, and C.
*****	Match *zero* or more occurrences of the preceding pattern.
+	Matches *one* or more occurrences of the preceding pattern.
?	Matches *zero or one* occurrences of the preceding pattern.
$	Matches end of line as the last character of the pattern.
{ }	Indicates how many times a pattern can be present. Example: A{1,3} implies one or three occurrences of A may be present.
****	Used to escape meta characters. Also used to remove the special meaning of characters as defined in this table.
^	Negation.
|	Logical OR between expressions.
"<some symbols>"	Literal meanings of characters. Meta characters hold.
/	Look ahead. Matches the preceding pattern only if followed by the succeeding expression. Example: A0/1 matches A0 only if A01 is the input.
( )	Groups a series of regular expressions.

4.4 Lex 代码分析

函数声明

```
%{
    inline bool printable_ascii(char ch);
    void yyerror(const char *message);
    void yyerror_unknownchar(char ch);
%}
%x COMMENT
```

定义部分，直接对该类型的按照一个正常的词法思路进行定义

```
whitespace  [\ \t\r]+
digit       [0-9]
alpha       [a-zA-Z]

hexdigit    [0-9A-Fa-f]
hexint      ${hexdigit}+

digits      {digit}+
exp         e[+-]?{digit}+

INTEGERT    {digits}
 /*|{hexint}*/
REALT       ({digits}\.{digits}?|{digits}?\.{digits}){exp}?
IDENTIFIER  [a-zA-Z][a-zA-Z0-9_$]*
STRINGT     \"([^"\n])*\"
BADSTR      \"([^"\n])*

COMMENTS    \(\*.*\*\)

LPAREN      \(
RPAREN      \)
LBRACKET    \[
RBRACKET    \]
LBRACE      \{
RBRACE      \}
COLON       :
DOT         \.
SEMICOLON ;
COMMA       ,
ASSIGN      :=
PLUS        \+
MINUS       -
STAR        \*
SLASH       \/
BACKSLASH \\
EQ          =
NEQ         <>
LT          <
LE          <=
GT          >
GE          >=
LABRACKET \[<
RABRACKET >\]

%%
```

匹配规则部分，下面会具体介绍

```
//空白符
{whitespace} { }
\n          { lineno++; }

//注释
```

```
 /* comment */
"(*"              { BEGIN(COMMENT); }
<COMMENT>[^)*\n]+
<COMMENT>\n        { lineno++; }
<COMMENT><<EOF>>    { yyerror("Unterminated comment"); return EOFF; }
<COMMENT>"*)"      { BEGIN(INITIAL); }
<COMMENT>[*)]

//符号规则匹配，写在定义里面了
{LPAREN}    { return LPAREN; }
{RPAREN}    { return RPAREN; }
{LBRACKET}  { return LBRACKET; }
{RBRACKET}  { return RBRACKET; }
{LBRACE}    { return LBRACE; }
{RBRACE}    { return RBRACE; }
{COLON}     { return COLON; }
{DOT}      { return DOT; }
{SEMICOLON} { return SEMICOLON; }
{COMMA}     { return COMMA; }
{ASSIGN}    { return ASSIGN; }
{PLUS}      { return PLUS; }
{MINUS}     { return MINUS; }
{STAR}      { return STAR; }
{SLASH}     { return SLASH; }
{BACKSLASH} { return BACKSLASH; }
{EQ}       { return EQ; }
{NEQ}      { return NEQ; }
{LT}       { return LT; }
{LE}       { return LE; }
{GT}       { return GT; }
{GE}       { return GE; }
{LABRACKET} { return LABRACKET; }
{RABRACKET} { return RABRACKET; }

PROGRAM     { return PROGRAM; }
IS         { return IS; }
BEGIN      { return BEGINT; }
END        { return END; }
VAR        { return VAR; }
TYPE       { return TYPE; }
PROCEDURE   { return PROCEDURE; }
ARRAY      { return ARRAY; }
RECORD     { return RECORD; }
IN         { return IN; }
OUT        { return OUT; }
READ       { return READ; }
WRITE      { return WRITE; }
IF         { return IF; }
THEN       { return THEN; }
ELSE       { return ELSE; }
ELSIF      { return ELSIF; }
WHILE      { return WHILE; }
DO         { return DO; }
LOOP       { return LOOP; }
FOR        { return FOR; }
EXIT       { return EXIT; }
```

```
RETURN     { return RETURN; }
TO         { return TO; }
BY         { return BY; }
AND        { return AND; }
OR         { return OR; }
NOT        { return NOT; }
OF         { return OF; }
DIV        { return DIV; }
MOD        { return MOD; }

//标识符
{IDENTIFIER}    {
    if(yyleng < 256)
        return IDENTIFIER;

    yyerror("Identifier too long");
    return ERROR;
}

//字符串常量判断，注意长度
{STRINGT}      {
    if(yyleng-2 < 256)
    {
        int i;
        for(i=1; i<yyleng-1; i++)
            if(!printable_ascii(yytext[i]))
                yyerror_unknownchar(yytext[i]);

        return STRINGT;
    }

    yyerror("String too long");
    return ERROR;
}

//整数常量判断，注意越界条件
{INTEGERT}     {
    if(yyleng <= 10)
    {
        int val = atoi(yytext);
        if(val >= 0)
            return INTEGERT;
    }

    yyerror("Integer out of range");
    return ERROR;
}

//实数常量，可直接转化
{REALT}        { return REALT; }

//未完结字符串
{BADSTR}       { yyerror("Unterminated string"); return ERROR; }

//文件结束
<<EOF>>        { return EOFF; }
```

```
//无法识别的字符
 /* fallback */
.             { yyerror_unknownchar(yytext[0]); return ERROR;}

%%

C 程序部分，下面是要用到的函数
inline int min(int a, int b)
{
    return (a < b) ? a : b;
}

//输出错误信息
void yyerror(const char *message)
{
    static char preview[14] = "??????????...";
    memcpy(preview, yytext, min(yyleng+1, 10));

    if(yytext)
        fprintf(stderr,"Error: \"%s\" in line %d. Token = `%s'\n",
           message, lineno, preview);
}

//未知字符错误
void yyerror_unknownchar(char ch)
{
    if(yytext)
        fprintf(stderr,"Error: \"Illegal character`\\%03o' ignored\"
in line %d.\n",
           ch, lineno);
}

//可打印字符
inline bool printable_ascii(char ch)
{
    return (0x20 <= ch && ch <= 0x7E && ch != '"');
}

//错误返回
int yywrap()
{
    return 1;
}
```