Compiler Principles Lab-1

wacky6 / Jiewei Qian (C)

This is the original lab report. I hope it can help you understanding wtf this (supposedly interesting but very badly-taught) lesson is talking about.

Copy with caution if your programming skill is weak or having no experience with non-VS environment, or your teacher will realize that you “learn” from someone else’s report.

: (

一、实验原理

大多数程序语言的单词符号可以用正规文法表述，通过正规文法可以构造对应的状态转移图，根据状态转移图就可以识别出单词和符号，从而实现词法分析程序。

实验要求实现类 C 语言的词法分析器：

1. 对单词分类，指出其类型。
2. 能够对程序进行预处理：去掉注释、多余空白
3. 读取文件处理

程序流程设计：

1. 读文件（通过 stdin）
2. 预处理，去掉注释，多余空白
3. 预处理检错（如/ /不匹配）
4. 词法分析
5. 词法分析检错

错误处理采用 SEH（结构化错误处理），分析器发现错误后 throw 错误信息，外层 catch 打印错误提示。利用 SEH，可以返回错误位置和错误信息，给出详细的错误原因。

因为是手写词法分析器，为了简化实现，程序仅实现了类 C 词法的子集。

预处理：

实现的语言支持/* comment */表示注释。在进行词法分析前，需要先将注释去除。方法很简单，从前到后扫描输入串，发现/*将注释状态置为 true，发现*/将注释状态置为 false。注释状态为 true 时，扫描到的字符不记录到输出。

因为/*长度为两个字符，所以程序内需要保存上一个字符的值，才能判断当前是否遇到了注释开始/结束标识。

词法分析：

要进行词法分析，首先要确定那些词是有意义的。先定义单词的构成和类型。这是词法分析的输出。

首先，确定程序语言的单词符号：

Tokens	Type
{	cbkt_start
}	cbkt_close
(	rbkt_start
)	rbkt_close
=	op_assign
==	op_equ
>=	op_gt_eq
>	op_gt
for	loop_for
break	loop_break
continue	loop_continue
return	return
if	flow_if
else	flow_else
func	function_decl
var	variable_decl
[identifier]	identifier
[integer]	integer
+	op_plus
-	op_minus
	op_multiply
/	op_div
,	coma
;	sem

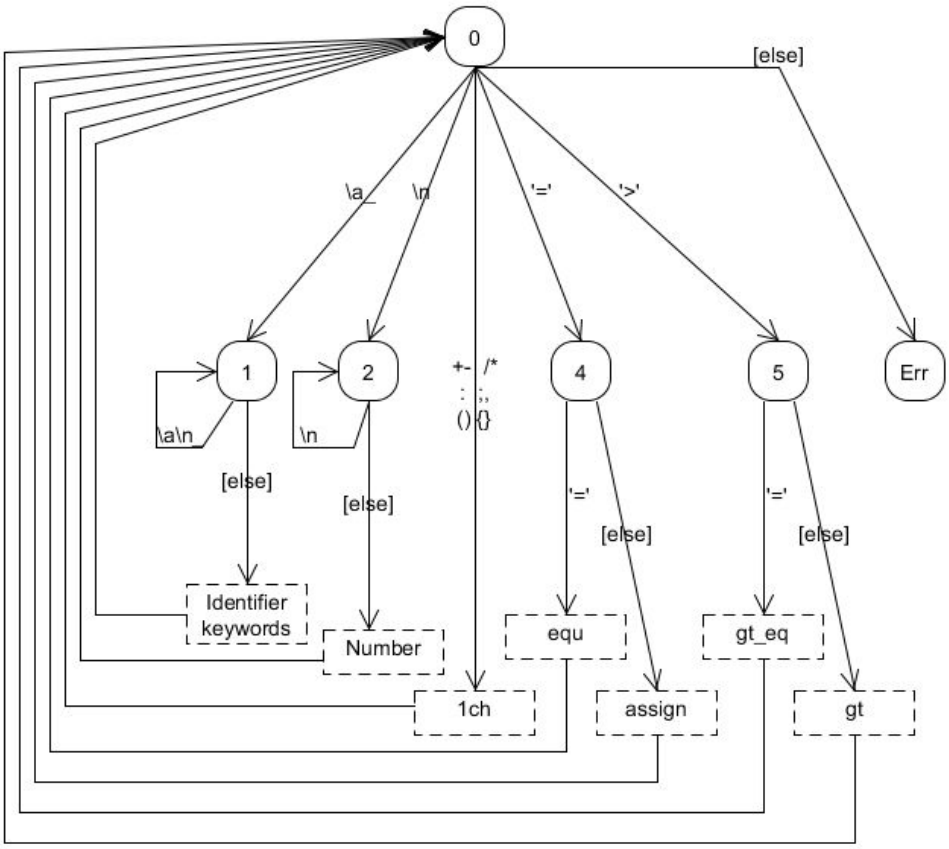
然后，我们按照单词符号的需求，写出正规（正则）表达式。这里使用了一些元字符集：\a 表示英文字母（[a-zA-Z]），\n 表示数字字符（[0-9]），<name>表示定义的一种类型。

对关键字（for、while）的处理：因为关键字的正则匹配是标识符匹配的子集，因此，在匹配到一个标识符时，将这个标识符和关键字一一比较：如果两者相同，就匹配到了一个关键字。利用这种处理方式，可以简化程序的实现。当然，将关键字识别整合到有限状态机中可是可行的，但在构造状态转移表时比较麻烦。在处理单字符符号时，也采用了这种方法。

```
<identifier> ::=  _\a|<identifier>\n|<identifier>\a
<integer>    ::=  \n|<integer>\n
<bkt>        ::=  {|}|(|)
<op_1ch>     ::=  +|-||/
<saperator>  ::=  ,|;
```

```
<1ch>       ::=  <op_1ch>|<saperator>|<bkt>
<eq>        ::=  =
<eq_x>      ::=  <eq>=
<gt>        ::=  >
<gt_x>      ::=  <gt>=
```

接着，根据正则表达式，推导出状态转移图。根据状态转移图，才能用有穷自动机的方式实现词法分析器。



最后，根据状态转换图，可以轻松地实现控制程序：

定义分析函数 F(S, C)。S 为当前状态，C 为当前字符。F 返回下一状态，如果 F 返回 Err，则说明输入的词法有错误。如果输入的词法是正确的，处理完输入后，状态 S 为 0。函数 F 在分析到语言符号时，会将符号记录下来，这是词法分析的输出。

程序代码：（省略了一些类型定义）

主程序：

```
int main(){
    string content = "";
    while (!cin.eof()) {
        string ln;
        std::getline(cin, ln);
```

```
        content = content + ln + " ";
    }
    try{
        content = preprocess(content);
    }catch(const char* s){
        cout<<"error: "<<s<<endl;
        return 0;
    }
    vector<token_info> tokens;
    try{
        tokens = lexer(content);
    }catch(const char* s){
        cout<<"error: "<<s<<endl;
        return 0;
    }
    for (size_t i=0, sz=tokens.size(); i!=sz; ++i)
        cout<<"  "<<left<<setw(12)<<tokens[i].type
            <<":  "<<tokens[i].token<<endl;
    return 0;
}
```

预处理：

```
string preprocess(const string& s) {
    string res;
    size_t i, sz;
    char   ch=0, last=0;
    bool   is_comment = false;
    res.reserve(s.length());
    for (i=0, sz=s.length(); i!=sz; ++i){
        last = ch;
        ch = s[i];
        if (last=='/' && ch=='*') {
            is_comment=true;
            res.pop_back();
        }
        if (last=='*' && ch=='/') {
            is_comment=false;
            continue;
        }
        if (is_comment) continue;
        if (isspace(last) && isspace(ch)) continue;
        res.push_back(ch);
    }
    if (is_comment)
        throw "matching */ not found";
```

```
    return res;
}
```

词法分析（有限状态机实现）：

```
vector<token_info> lexer(const string& s) {
    vector<token_info> tokens;
    size_t i, sz;
    i=0; sz=s.length();
    #define GETCH() (i==sz ? '\0' : s[i++])
    char ch;  // current character
    size_t state = 0;
    string token = "";
    static char msg[512];  // used to throw error
    #define ST(_state, _append) {T(_append); S(_state);}
    #define S(_state) {state=_state; break;}
    #define T(_append) {token+=_append;}
    #define TOKEN(_T) {tokens.push_back(TOKEN_INFO(_T, token)); token=""; state=0;}
    do{
        ch = GETCH();  // on EOF: \0, terminate previous token
        cerr<<"state = "<<state<<", token="<<token<<" ch='"<<ch<<"'"<<endl;
        switch(state){
            case 0:
                if (is_alpha_uscr(ch)) ST(1,ch);
                if (isnumber(ch))      ST(2,ch);
                if (ch=='=')           ST(3,ch);
                if (ch=='>')           ST(4,ch);
                if (isspace(ch))       S(0);
                // 1 chacacter token
                T(ch)
                if (ch=='(') TOKEN(192);
                if (ch==')') TOKEN(193);
                if (ch=='{') TOKEN(194);
                if (ch=='}') TOKEN(195);
                if (ch==',') TOKEN(209);
                if (ch==';') TOKEN(208);
                if (ch=='\0') continue; // \0 marks EOF
                if (is_binary_op(ch)) TOKEN(200);
                if (token.length()) { // not a valid 1-char token
                    i--;
                    S(127);
                }
            break;
            case 1:
```

```
                if (is_alpha_uscr(ch) || isnumber(ch)) ST(1,ch);
                TOKEN(identify_keywords(token)); // identifier
                i--;
            break;
            case 2:
                if (isnumber(ch)) ST(2,ch);
                TOKEN(129); // number
                i--;
            break;
            case 3:
                if (ch=='=') {
                    T(ch);
                    TOKEN(200);    // op_binary: equ
                }else if (ch=='>') {
                    T(ch);
                    TOKEN(240);    // lambda
                }else {
                    TOKEN(224);    // op_assign
                    i--;
                }
            break;
            case 4:
                if (ch=='=') {
                    T(ch);
                    TOKEN(200);   // gt_eq
                }else{
                    TOKEN(200);
                    i--;      // gt
                }
            break;
            case 127: // unexpected token
                sprintf(msg, "unexpected token: %c", ch);
                throw msg;
            break;
            default:
                // this should not happen.
                throw "unknown error! FSM enters an undefined state!!";
            break;
        }
    }while(ch);
    return tokens;
}
```

二、实验小结

词法分析是编译器的基础，它识别出一种语言的单词。单词是语言语义的基础，是语言的组成单位。

通过词法分析程序的设计，复习了有限状态机的概念，了解了有限状态机在编译器实现中的重要应用。

目前，有一些开源软件可以根据语法自动生成词法分析、语法分析程序，比如 bison、flex 等，这些工具简化了编译器的开发过程。

对一个语言：字符 > 单词 > 语法 > 语义。在进行编译器设计的时候，一般也遵从这个结构。