厦門大學

软件学院

《编译技术》实验报告

题　　目　语法制导的三地址代码生成程序

姓　　名　陈澄

学　　号　32420212202930

班　　级　软工三班

实验时间　2024/05/14

2024 年 **05** 月 **14** 日

1 实验目的

掌握计算机语言的语法分析程序设计与属性文法应用的实现方法。

2 实验环境

编写语言：C++

编译环境：Visual Studio2022

3 实验内容

编制一个能够进行语法分析并生成三地址代码的微型编译程序。

4 实验步骤

1、考虑给定的文法，消除左递归，提取左因子。

给定文法 G 如下：

S -> id = E ;

S -> if C then S1 ;

S -> if C then S1 else S2 ;

S -> while C do S1 ;

C -> E1 > E2

C -> E1 < E2

C -> E1 = E2

E -> E1 + T

E -> E1 - T

E -> T

T -> F

T -> T1 * F

T -> T1 / F

F -> (E)

F -> id

F -> int8

F -> int10

F -> int 16

其中仅标红部分有左递归，消除左递归，提取左因子后文法 G 如下：

S -> id = E ;

S -> if C then S S';

S' -> ε

S' -> else S

S -> while C do S ;

C -> E C'

C' -> > E

C' -> < E

C' -> = E

E -> T E'

E' -> + E E'

E' -> - E E'

E' -> ε

T -> F T'

T' -> * F T'

T’ -> / F T’

T’ -> ε

F -> (E)

F -> id

F -> int8

F -> int10

F -> int 16

2、编制并化简语法图。

FIRST(S) = {id , if , while}

FIRST(S’) = {else , ε}

FOLLOW(S) = {else , ; , #}

FOLLOW(S’) = {else , ; , #}

FIRST(C) = {(, id , int 8 , int10 , int16}

FIRST(C’) = {> , < , =}

FOLLOW(C) = {do , then}

FOLLOW(C’) = {do , then}

FIRST(E) = {(, id , int 8 , int10 , int16}

FIRST(E′) = {+ , - , ε}

FIRST(T) = {(, id , int 8 , int10 , int16}

FIRST(T′) = {* , / , ε}

FIRST(F) = {(, id , int 8 , int10 , int16}

FOLLOW(E) = {) , # , > , < , = , ;}

FOLLOW(E′) = {) , # , > , < , = , ;}

FOLLOW(T) = {+ , - ,) , # , > , < , = , ;}

FOLLOW(T′) = {+ , - ,) , # , > , < , = , ;}

FOLLOW(F) = {* , / , + , - ,) , # , > , < , = , ;}

假设 then 匹配最近的 else 则满足 LL(1)文法。

语法图（预测分析表）：

	id	if	else	while	其他	#
S	S-> id = E	S-> if C then S S'		S-> while C do S		
S’			S'-> else S			

	>	<	=	其他
C	C-> E C'	C-> E C'	C-> E C'	C-> E C'
C’	C'-> +|-|= E	C'-> +|-|= E	C'-> +|-|= E	

	+	-	*	/	其他
E	E-> T E'	E-> T E'	E-> T E'	E-> T E'	E-> T E'
E’	E'-> +|- E E'	E'-> +|- E E'			
T	T-> F T'	T-> F T'	T-> F T'	T-> F T'	T-> F T'
T’			T'-> *|/ F T'	T'-> *|/ F T'	

	(	idn | int8 | int10 | int16
F	F-> E	F-> id | int8 | int10 | int16

3、编制递归子程序的算法。

```
void matchToken(int expected) {
   if (lookahead != expected) {
      err();
   }
   else {
      lookahead = getToken();
   }
}
Void 某非终结符() {
   if(lookahead == token1) {
      matchToken(token1);
      ...(按照产生式匹配后续 token)
   }
   else if(lookahead == token2) {
      ....;
   }
}
```

4、编制各个递归子程序函数。

matchToken()方法如下：

```
void matchToken(int expected) {
    if (lookahead != expected) {
        err();
    }
    else {
        lookahead = getToken();
    }
}
```

各个递归子程序函数：

S：

```
void S() {
    if (lookahead == idn) {
        cout << "S-> id = E" << endl;
        matchToken(idn);
        matchToken(equal);
        E();
        matchToken(sem);
    }
    else if (lookahead == while_tkn) {
        cout << "S-> while C do S S'" << endl;
        matchToken(while_tkn);
        C();
        matchToken(do_tkn);
        S();
        matchToken(sem);
    }
    else if (lookahead == if_tkn) {
        cout << "S-> if C then S S'" << endl;
        matchToken(if_tkn);
        C();
        matchToken(then_tkn);
        S();
        S_();
        matchToken(sem);
    }
    else err();
}
```

S’：

```
void S_() {
    if (lookahead == else_tkn) {
        cout << "S'-> else S" << endl;
        matchToken(else_tkn);
        S();
    }
    else return;
}
```

C：

```
void C() {
    cout << "C-> E C'" << endl;
    E();
    C_();
}
```

C’：

```
void C_() {
    if (lookahead == greater || lookahead == less || lookahead == equal) {
        cout << "C'-> +|-|= E" << endl;
        matchToken(lookahead);
        E();
    }
    else err();
}
```

E：

```
void E() {
    cout << "E-> T E'" << endl;
    T();
    E_();
}
```

E’：

```
void E_() {
    if (lookahead == add || lookahead == sub) {
        cout << "E'-> +|- E E'" << endl;
        matchToken(lookahead);
        E();
        E_();
    }
    else return;
}
```

T：

```
void T() {
    cout << "T-> F T'" << endl;
    F();
    T_();
}
```

T’：

```
void T_() {
    if (lookahead == mul || lookahead == div) {
        cout << "T'-> *|/ F T'" << endl;
        matchToken(lookahead);
        F();
        T_();
    }
    else return;
}
```

F：

```
void F() {
    if (lookahead == l_par) {
        cout << "F-> (E)" << endl;
        matchToken(l_par);
        E();
        matchToken(r_par);
    }
    else if (lookahead == idn) {
        cout << "F-> idn" << endl;
        matchToken(idn);
    }
    else if (lookahead == int8) {
        cout << "F-> idn8" << endl;
        matchToken(int8);
    }
    else if (lookahead == int10) {
        cout << "F-> idn10" << endl;
        matchToken(int10);
    }
    else if (lookahead == int16) {
        cout << "F-> idn16" << endl;
        matchToken(int16);
    }
    else err();
}
```

5、连接实验一的词法分析函数 scan( )，进行测试。

连接实验一的词法分析器编写 getToken()方法：

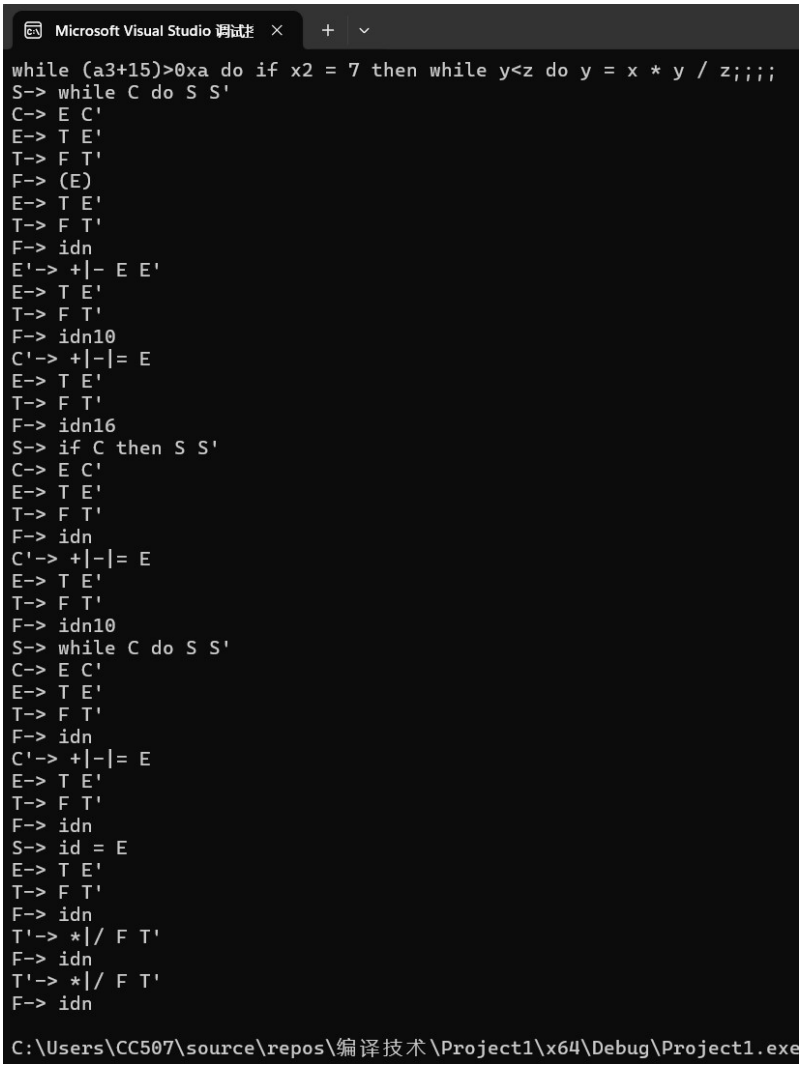
```
int scan() { ... }
int processStatus(int status) {
    switch (status) { ... }
    return 0;
}//将状态处理为对应属性
int getToken() {
    while (s[0] == ' ')s++;//每次获得token前先去除空格
    return processStatus(scan());
}
```

token 定义：

```
#define idn 1
#define int8 2
#define int10 3
#define int16 4
#define if_tkn 5
#define while_tkn 6
#define else_tkn 7
#define then_tkn 8
#define do_tkn 9
```

```
#define add 10
#define sub 11
#define mul 12
#define div 13
#define less 14
#define greater 15
#define equal 16
#define l_par 17
#define r_par 18
#define sem 19
```

试运行结果：



可见语法分析没有问题，但是原语法有匹配多个分号的问题，故将原语法改为：

S0 -> S;

S -> id = E

S -> if C then S S’

S’ -> ε

S’ -> else S

S -> while C do S

开始符号改为 S0

6、设计三地址代码生成的数据结构和算法。

数据结构如下设计：

```
class S{
public:
    string place, code, begin, next;
};
class C {
public:
    string true_, false_, code;
};
typedef class {
public:
    string code, place;
}E, T, F;
```

函数名以及传参修改：

```
S parse_S0(S);
S parse_S(S);
S parse_S_(C&,S);
C parse_C();
C parse_C_(E);
E parse_E();
E parse_E_(E);
F parse_F();
T parse_T();
F parse_T_(T);
```

算法：

E,T,F 中 place 存储当前变量名，code 存储当前代码

算术表达式计算时 place 作为之前所有运算得到的值，并传入下一个递归子程序与下一个变量进行运算，最终传出并生成三地址代码存储在 code 中。

C 中 true_、false_存储当前布尔表达式的跳转标号，code 存储当前代码

布尔表达式计算时 true_存入 code 时先使用“!”进行填入，等到整个语句分析完后再进行回填，false_同理使用“?”先行填入。

S 中 begin 存储当前语句开始标号，next 存储下一语句标号，code 存储当前代码。

在递归子程序中生成的三地址代码会累加到 code 中最终返回。

7、 将各个递归子程序函数改写为代码生成函数。

为各个递归子程序加入语义分析：

```
S parse_S(S s) {
    S s1;
    if (lookahead == idn) {
        //cout << "S-> id = E" << endl;
        string id = lookaheadstring;
        matchToken(idn);
        matchToken(equal);
        E e = parse_E();
        s1.code = e.code + id + " := " + e.place + "\n";
    }
    else if (lookahead == while_tkn) {
        //cout << "S-> while C do S" << endl;
        matchToken(while_tkn);
        C c = parse_C();
        if (s.begin == "")s1.begin = newlabel();
        else s1.begin = s.begin;
        c.true_ = newlabel();
        c.false_ = s.next;
        c.code.replace(c.code.find('!'), 1, c.true_);
        c.code.replace(c.code.find('?'), 1, c.false_);
        matchToken(do_tkn);
        s1.next = s.begin;
        S s2 = parse_S(s1);
        s2.next = s.begin;
        s1.code = ((s.begin == "") ? (s1.begin + ": ") : "")
            + c.code + c.true_ + ": " + s2.code + "goto: " + s1.begin + "\n";
    }
    else if (lookahead == if_tkn) {
        //cout << "S-> if C then S S'" << endl;
        matchToken(if_tkn);
        C c = parse_C();
        c.true_ = newlabel();
        c.false_ = s.next;
        c.code.replace(c.code.find('!'), 1, c.true_);
        matchToken(then_tkn);
        s1.next = s.next;
        s1.begin = c.true_;
        S s2 = parse_S(s1);
        S s3 = parse_S_(c,s1);
        if (s3.code == "")c.code.replace(c.code.find('?'), 1, c.false_);
        s2.next = s.next;
        s3.next = s.next;
        s1.code = c.code + c.true_ + ": " + s2.code + s3.code;
    }
    else err();
    return s1;
}
```

```
S parse_S_(C& c,S s) {
    S s1;
    if (lookahead == else_tkn) {
        //cout << "S'-> else S" << endl;
        matchToken(else_tkn);
        S s2 = parse_S(s1);
        s1.next = s.next;
        c.false_ = newlabel();
        c.code.replace(c.code.find('?'), 1, c.false_);
        s1.code = "goto: " + s.next + "\n" + c.false_ + ": " + s2.code;
    }
    else return s1;
}
```

```
C parse_C() {
    //cout << "C-> E C'" << endl;
    E e = parse_E();
    return parse_C_(e);
}
```

```
C parse_C_(E e) {
    C c1;
    if (lookahead == greater) {
        //cout << "C'-> > E" << endl;
        matchToken(lookahead);
        E e1 = parse_E();
        c1.code = e.code + e1.code + "if " + e.place + " > " + e1.place + " goto: !\ngoto: ?\n";
    }
    else if (lookahead == less) {
        //cout << "C'-> < E" << endl;
        matchToken(lookahead);
        E e1 = parse_E();
        c1.code = e.code + e1.code + "if " + e.place + " < " + e1.place + " goto: !\ngoto: ?\n";
    }
    else if (lookahead == equal) {
        //cout << "C'-> = E" << endl;
        matchToken(lookahead);
        E e1 = parse_E();
        c1.code = e.code + e1.code + "if " + e.place + " = " + e1.place + " goto: !\ngoto: ?\n";
    }
    else err();
    return c1;
}
```

```
E parse_E() {
    //cout << "E-> T E'" << endl;
    T t = parse_T();
    return parse_E_(t);
}
```

```
E parse_E_(E e) {
    E e1;
    if (lookahead == add) {
        //cout << "E'-> + E E'" << endl;
        matchToken(add);
        E e2 = parse_E();
        e1.place = newtemp();
        e1.code = e.code + e2.code + e1.place + " := " + e.place + " + " + e2.place + "\n";
        return parse_E_(e1);
    }
    else if (lookahead == sub) {
        //cout << "E'-> - E E'" << endl;
        matchToken(sub);
        E e2 = parse_E();
        e1.place = newtemp();
        e1.code = e.code + e2.code + e1.place + " := " + e.place + " - " + e2.place + "\n";
        return parse_E_(e1);
    }
    else {
        return e;
    }
}
```

```
T parse_T() {
    //cout << "T-> F T'" << endl;
    F f = parse_F();
    return parse_T_(f);
}
```

```
T parse_T_(T t) {
    T t1;
    if (lookahead == mul) {
        //cout << "T'-> * F T'" << endl;
        matchToken(mul);
        F f = parse_F();
        t1.place = newtemp();
        t1.code = t.code + f.code + t1.place + " := " + t.place + " * " + f.place + "\n";
        return parse_T_(t1);
    }
    else if (lookahead == div) {
        //cout << "T'-> / F T'" << endl;
        matchToken(div);
        F f = parse_F();
        t1.place = newtemp();
        t1.code = t.code + f.code + t1.place + " := " + t.place + " / " + f.place + "\n";
        return parse_T_(t1);
    }
    else {
        return t;
    }
}
```

```
F parse_F() {
    F f;
    if (lookahead == l_par) {
        //cout << "F-> (E)" << endl;
        matchToken(l_par);
        E e = parse_E();
        matchToken(r_par);
        f.place = e.place;
        f.code = e.code;
    }
    else if (lookahead == idn) {
        //cout << "F-> idn" << endl;
        f.place = lookaheadstring;
        f.code = "";
        matchToken(idn);
    }
    else if (lookahead == int8) {
        //cout << "F-> idn8" << endl;
        f.place = to_string(stoi(lookaheadstring.substr(2), nullptr, 8));
        f.code = "";
        matchToken(int8);
    }
    else if (lookahead == int10) {
        //cout << "F-> idn10" << endl;
        f.place = lookaheadstring;
        f.code = "";
        matchToken(int10);
    }
    else if (lookahead == int16) {
        //cout << "F-> idn16" << endl;
        f.place = to_string(stoi(lookaheadstring.substr(2), nullptr, 16));
        f.code = "";
        matchToken(int16);
    }
    else err();
    return f;
```

```
S parse_S0(S s) {
    S s1 = parse_S(s);
    matchToken(sem);
    return s1;
}
```

8、编制测试程序（main 函数）。

main 函数中读取输入串，

获取第一个 token,

调用 parse_S0()进行三地址代码生成并输出。

```
int main() {
    string input;
    getline(cin, input);
    s = (char*)input.c_str();
    lookahead = getToken();
    S s;
    s.next = newlabel();
    s.begin = "";
    s.code = "";
    cout << parse_S0(s).code;
    cout << "L0:  //S.next";
}
```

9、调试程序：输入一个语句，检查输出的三地址代码 。

```
Microsoft Visual Studio 调试控
while (a3+15)>0xa do if x2 = 7 then while y<z do y = x * y / z;
L1: t1 := a3 + 15
if t1 > 10 goto: L2
goto: L0
L2: if x2 = 7 goto: L3
goto: L1
L3: if y < z goto: L4
goto: L1
L4: t2 := x * y
t3 := t2 / z
y := t3
goto: L3
goto: L1
L0:  //S.next
C:\Users\CC507\source\repos\编译技术\Project1\x64\Debug\Project1.ex
按任意键关闭此窗口. . .
```

5 思考题

1. 生成的三地址代码可否直接输出（不采用数据结构来实现属性 code）？

答：

不可直接输出，因为条件语句涉及代码的标号问题，跳转的标号需要整条语句识别完整之后才能确定，如果直接输出会导致无法确定标号。

2. 如何保证四则运算的优先关系和左结合性？

答：

优先关系：

文法中 T 代表乘除运算，E 为加减运算，F 为括号运算，文法中 F 的递归深度大于 T 的递归深度大于 E 意味着识别时会优先识别括号然后是乘除最后才进行加减。

左结合性：

匹配 token 时从左到右识别即可保证左结合性。

3． 如何采用代码段相对地址代替三地址代码序列中的标号？

答：

遍历一遍生成的三地址代码，按顺序为每条代码生成一个相对地址即可。