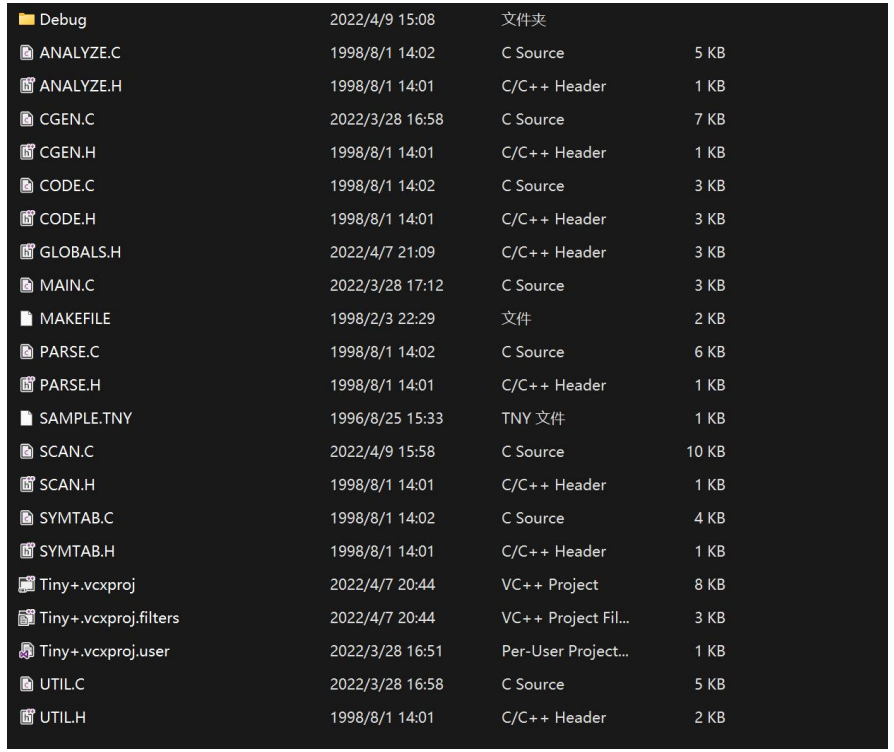
编译原理语法分析报告

陆逸凡

2019141090163

1.编程语言及开发环境

- 编程语言：C语言
- 开发环境：
  - 硬件：处理器AMD Ryzen 7 5800H with Radeon Graphics　　3.20 GHz
  - 软件：Windows11 家庭版
  - IDE: Microsoft Visual Studio 2019



Debug	2022/4/9 15:08	文件夹	
ANALYZE.C	1998/8/1 14:02	C Source	5 KB
ANALYZE.H	1998/8/1 14:01	C/C++ Header	1 KB
CGEN.C	2022/3/28 16:58	C Source	7 KB
CGEN.H	1998/8/1 14:01	C/C++ Header	1 KB
CODE.C	1998/8/1 14:02	C Source	3 KB
CODE.H	1998/8/1 14:01	C/C++ Header	3 KB
GLOBALS.H	2022/4/7 21:09	C/C++ Header	3 KB
MAIN.C	2022/3/28 17:12	C Source	3 KB
MAKEFILE	1998/2/3 22:29	文件	2 KB
PARSE.C	1998/8/1 14:02	C Source	6 KB
PARSE.H	1998/8/1 14:01	C/C++ Header	1 KB
SAMPLE.TNY	1996/8/25 15:33	TNY 文件	1 KB
SCAN.C	2022/4/9 15:58	C Source	10 KB
SCAN.H	1998/8/1 14:01	C/C++ Header	1 KB
SYMTAB.C	1998/8/1 14:02	C Source	4 KB
SYMTAB.H	1998/8/1 14:01	C/C++ Header	1 KB
Tiny+.vcxproj	2022/4/7 20:44	VC++ Project	8 KB
Tiny+.vcxproj.filters	2022/4/7 20:44	VC++ Project Fil...	3 KB
Tiny+.vcxproj.user	2022/3/28 16:51	Per-User Project...	1 KB
UTIL.C	2022/3/28 16:58	C Source	5 KB
UTIL.H	1998/8/1 14:01	C/C++ Header	2 KB

2. 变量声明语句的实现

2.1 定义单变量

定义多个单变量的语句形如：int x, y, z;

EBNF推导如下：

define_stmt→ type ID {COMMA ID} SEMI

| type ID EQ NUM

type→ INT | FLOAT

使用**递归下降分析**。增加id_seq来表述id序列。
修改后的EBNF如下：

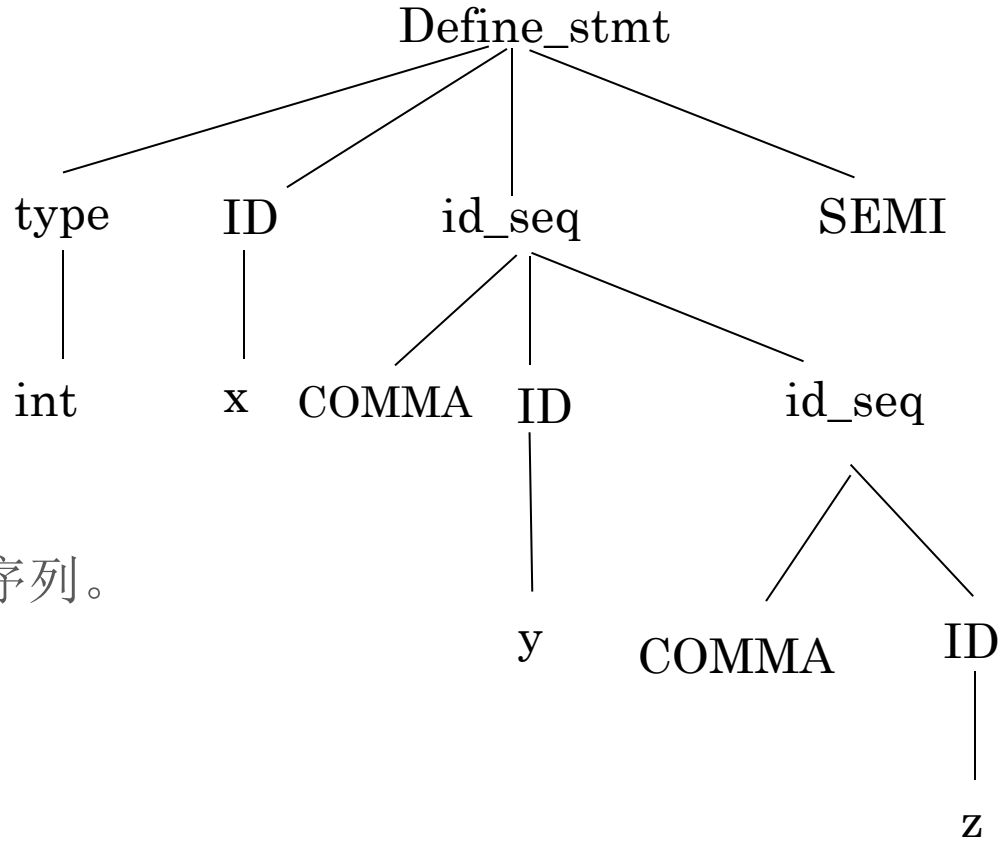
define_stmt→ type ID id_seq SEMI

| type ID EQ NUM

id_seq→ {COMMA ID}

type→ INT | FLOAT

语法树如下：



2.1 定义单变量

在parcer.c文件中添加两个函数 define_stmt()以及id_seq()来实现功能。

```
/*新添的变量声明语句*/
TreeNode* define_stmt(void)
{
    TreeNode* t = newStmtNode(DefineK);
    switch (token)
    {
    case INT:
        match(INT);
        break;
…
    }

    TreeNode* p = factor();
    t->child[0] = p;

    else if(t!=NULL && token==COMMA)
    {

        t->child[1] = ID_seq();//int x, y, z;
    }
…
    return t;
}
```

```
/*新添的ID序列语句*/
TreeNode* ID_seq(void)
{
    //TreeNode* t = newSmtNode(IdSeqK);
    TreeNode* t = NULL;

    while (token == COMMA)
    {
        match(COMMA);
        TreeNode* q = newStmtNode(IdSeqK);
        TreeNode* p = factor();
        if (q!=NULL)
        {
            q->child[0] = p;
            q->child[1] = ID_seq();
            t = q;

}


    }
    return t;
}
```

2. 变量声明语句的实现

2.1 定义单变量

在定义单变量时进行赋值形如：int x = 1;

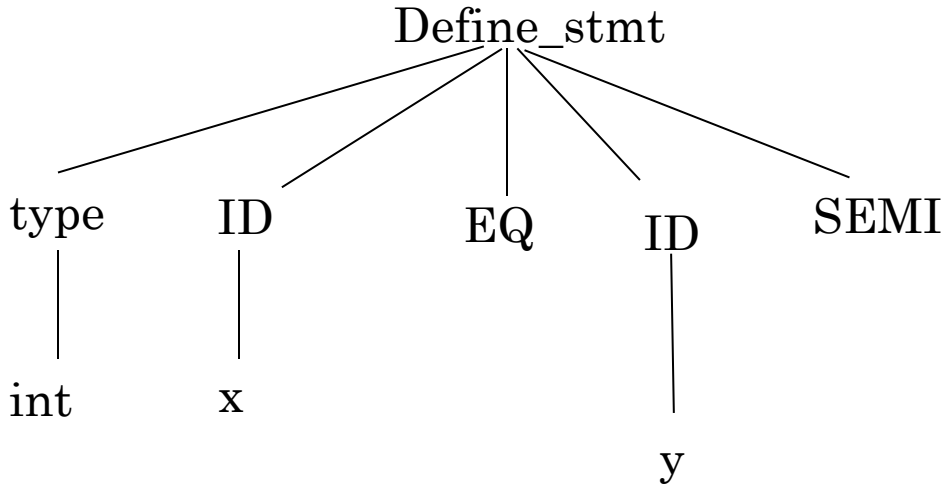
EBNF推导如下：

define_stmt→ type ID EQ NUM SEMI

type→ INT | FLOAT

使用**递归下降分析**。

语法树如下：



2.1 定义单变量

在define_stmt()中添加检测。

```
/*新添的变量声明语句*/
TreeNode* define_stmt(void)
{
    TreeNode* t = newStmtNode(DefineK);
    switch (token)
    {
    case INT:
        match(INT);
        break;
...
    }

    TreeNode* p = factor();
    t->child[0] = p;

     if (t != NULL && token == EQ)
    {
        match(EQ);
        t->child[1] = exp(); // int x = 1;
    }

...
    return t;
}
```

2.1 定义单变量

编写测试文件：

```
int x = 1;
int a,b;
int a,b,c;
int q,w,e,r,t;

float x = 1;
float a,b;
float a,b,c;
```

测试结果：

```
Syntax tree:
  Define:
    Id: x
    Const: 1
  Define:
    Id: a
    Define sequence:
      Id: b
  Define:
    Id: a
    Define sequence:
      Id: b
      Define sequence:
        Id: c
  Define:
    Id: q
    Define sequence:
      Id: w
      Define sequence:
        Id: e
        Define sequence:
          Id: r
          Define sequence:
            Id: t
  Define:
    Id: x
    Const: 1
  Define:
    Id: a
    Define sequence:
      Id: b
  Define:
    Id: a
    Define sequence:
      Id: b
      Define sequence:
        Id: c
```

2.2 定义多维数组

一维数组的BCNF推导需要在之前的define_stmt推导中添加如下语句：

define_stmt→ type ID LBRA NUM RBRA SEMI

Type→ INT | FLOAT

为了定义多元数组： int a[2][3]，还需要进行如下修改：

define_stmt→ type ID {LBRA NUM RBRA}

使用**递归下降分析**。增加id_seq来表述id序列。修改后的EBNF如下：

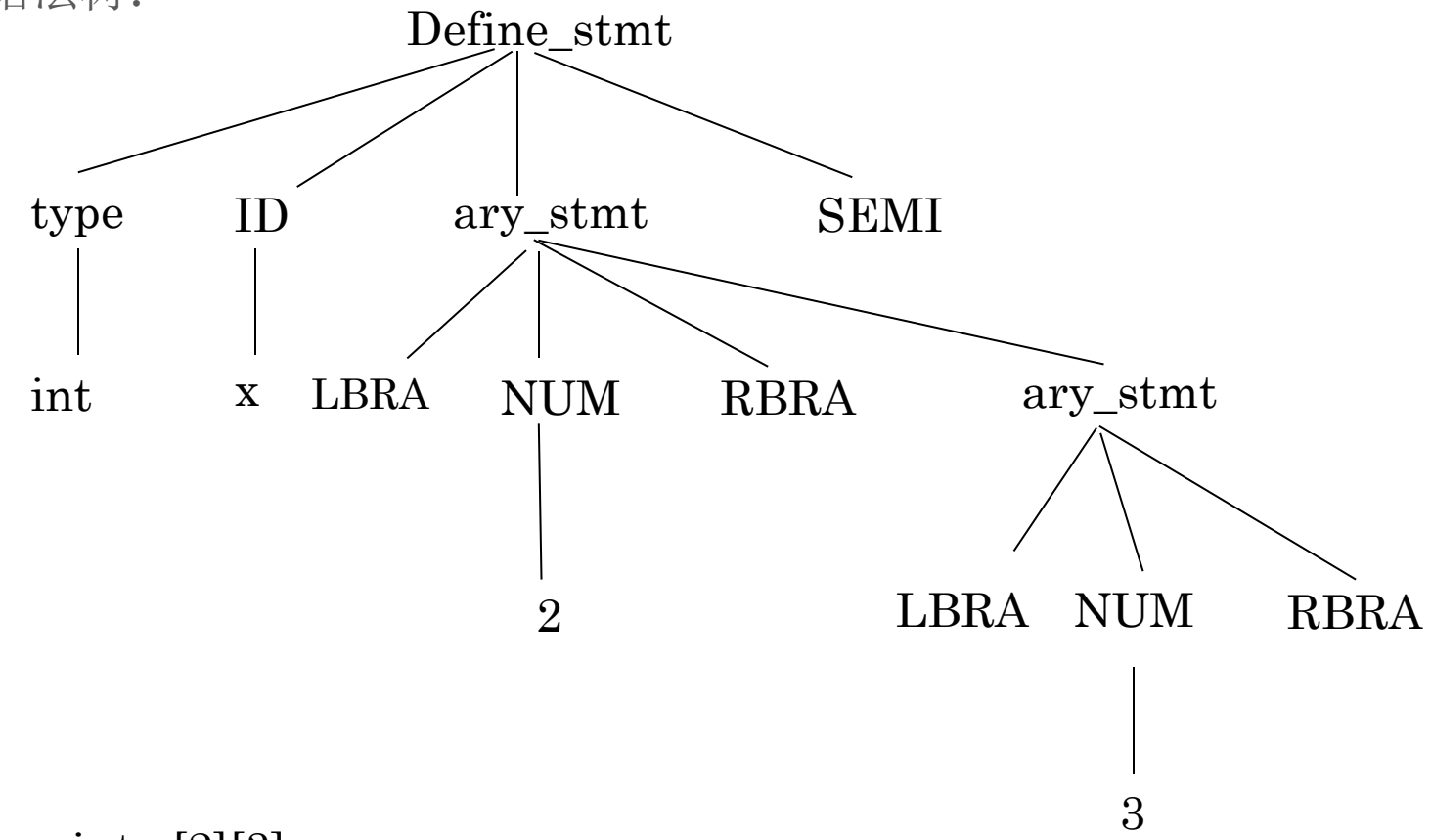
define_stmt→ type ID ary_stmt SEMI

Ary_stmt→ {LBRA ID RBRA}

type→ INT | FLOAT

2.2 定义多维数组

数组的语法树：



int x[2][3]

2.2 定义多维数组

在函数 define_stmt()中添加对“[”的识别，以及增加ary_stmt()来实现多维数组下表识别和添加功能。

```
/*新添的变量声明语句*/
TreeNode* define_stmt(void)
{
    TreeNode* t = newStmtNode(DefineK);
    switch (token)
    {
    case INT:
        match(INT);
        break;
…
    }

    TreeNode* p = newExpNode(IdK);
    p->attr.name = copyString(tokenString);
    t->child[0] = p;
    match(ID);
    else if(t!=NULL && token==LBRA)
    {

        t->child[1] = ary_stmt();//int
a[2][3];
    }
…
    return t;
}
```

```
/*新添的数组下标语句*/
TreeNode* ary_stmt(void)
{
    TreeNode* t = NULL;
    while (token == LBRA)
    {
        match(LBRA);
        TreeNode* q = newStmtNode(AryK);
        TreeNode* p = newExpNode(ConstK);
        p->attr.val = atoi(tokenString);
        match(NUM);
        match(RBRA);
        if (q != NULL)
        {
            q->child[0] = p;
            q->child[1] = ary_stmt();
            t = q;
        }

    }
    return t;
}
```

2.3 使数组元素成为表达式运算分量

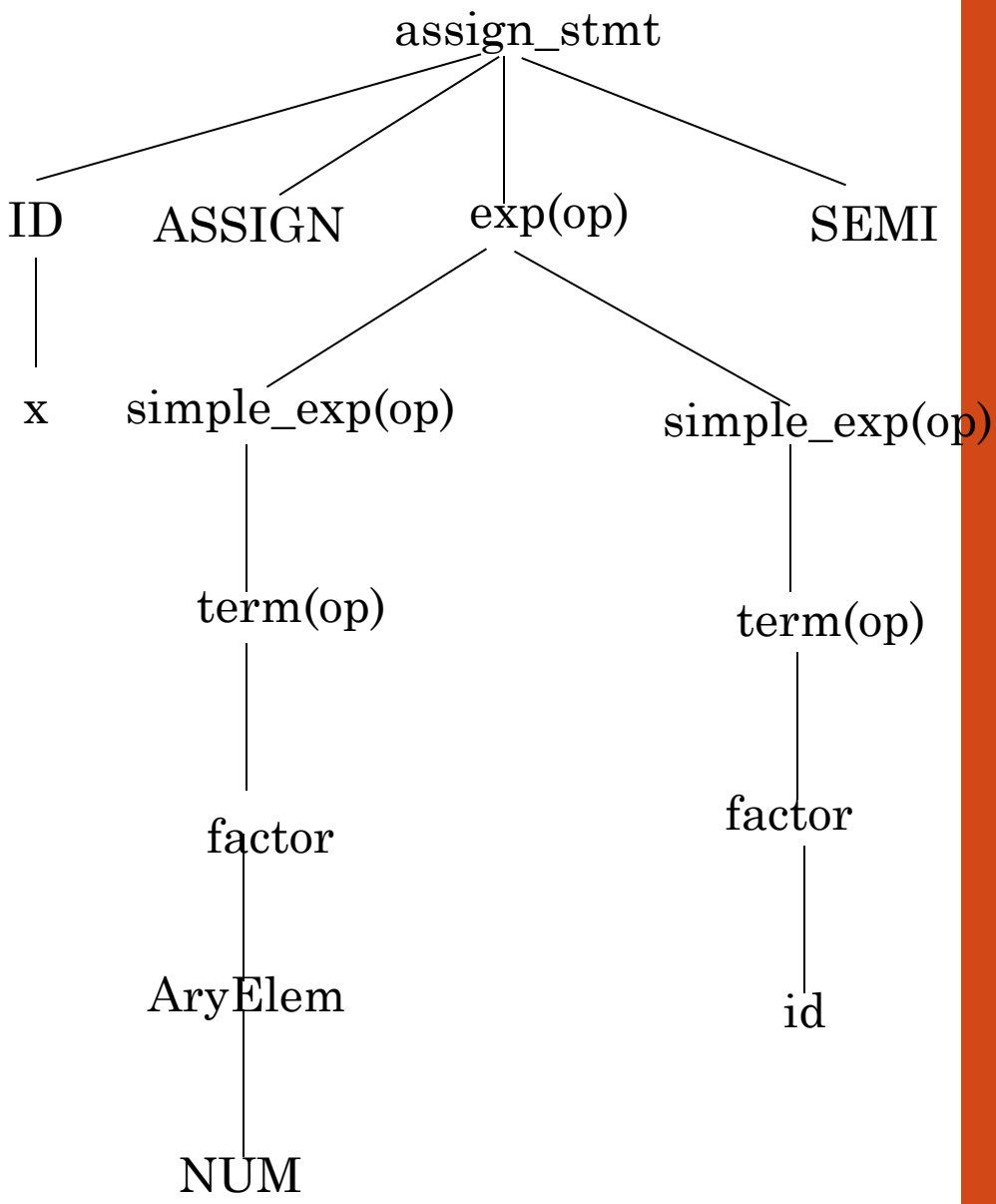
为了使数组元素成为表达式的运算分量，需要在factor()中添加相应的识别过程。

首先添加新的ExpKind类型：AryEleK，用来描述一个数组的元素。

```
typedef enum {OpK,ConstK,IdK, FunCallK, AryEleK} ExpKind;
```

在factor识别到ID后，检查下一token是否为“[”，若是则新建数组元素节点。

包括一个一维数组元素为运算分量的表达式的语法树如右图。



2.2 定义多维数组

编写测试文件：

```
1  int a[2];
2  int b[3][2];
3
4  x := a[2] + 0;
5
```

测试结果：

```
Syntax tree:
  Define:
    Id: a
    Index number:
      Const: 2
  Define:
    Id: b
    Index number:
      Const: 3
      Index number:
        Const: 2
  Assign to: x
    Op: +
      Array Element: a, index:
        Const: 2
      Const: 0
```

分别测试了：

1.一维数组的声明

2.多维数组的声明

3.将数组元素作为表达式的运算分量

4.声明语句在程序中出现的位置自定义

3. 函数的实现

3.1 函数定义

函数的定义EBNF推导如下：

Function→ return_type ID LPAREN [param_seq] RPAREN block

param_seq→ type_seq

type_seq→ type ID {COMMA type ID}

block→ BEGIN stmt_sequence END

stmt_seq→ RETURN exp

|....

Return_type→ type

3. 函数的实现

3.1 函数定义

进行如下的修改：

- 新建函数类型节点FuncK；
- 新建两种函数类型枚举：MainK、NormK代表主函数和其他函数；
- 新建StmtKind枚举PramK（参数列表节点）、ReturnK（函数返回语句节点）、TypeSeqK（参数类型列表节点）；
- 新建ExpType：Return（函数返回值类型）

3.2 函数定义

使用一系列函数来组成整个程序，因此需要将原本的stmt_sequence改为func_sequence，并将stmt_sequence嵌入各函数的主体中。

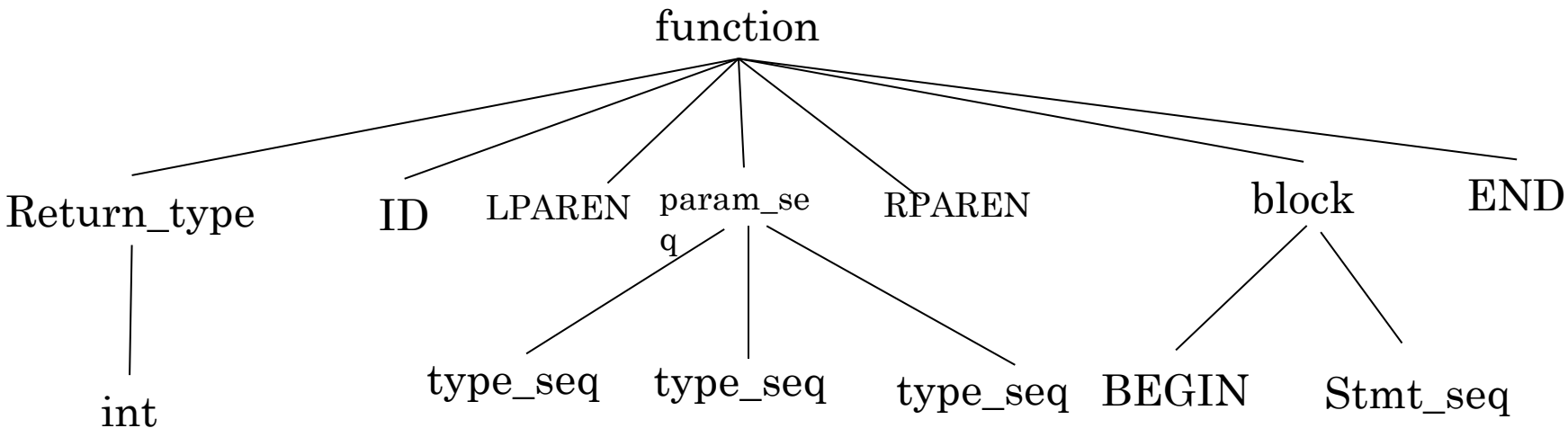
```
//程序由多个函数组成
TreeNode* func_sequence(void)
{
    TreeNode* t = function();
    TreeNode* p = t;
    while (token != ENDFILE )
    {
        TreeNode* q;
        q = function();
        if (q != NULL) {
            if (t == NULL) t = p = q;
            else {
                p->sibling = q;
                p = q;
            }
        }
    }
    return t;
}
```

```
TreeNode * parse(void)
{ TreeNode * t;
  token = getToken();
  //t = stmt_sequence();
  t = func_sequence();
  if (token!=ENDFILE)
    syntaxError("Code ends before
file\n");
  return t;
}
```

修改parse()函数

3.2 函数定义

一个函数的语法树如下：



3.2 函数调用

在ExpKind中新增函数调用结点：FunCallK

在factor中编写识别函数调用语句以实现函数作为表达式额运算分量。

3. 函数

编写测试文件：

```
1  int fun(int a)
2  begin
3      return a+1
4  end
5
6  int min(int a, int b)
7  begin
8      int temp=0;
9      return a
10 end
11
12 int main()
13 begin
14     int x = 1;
15     int a;
16     a := fun(x);
17     return x
18 end
```

普通函数，含一个参数

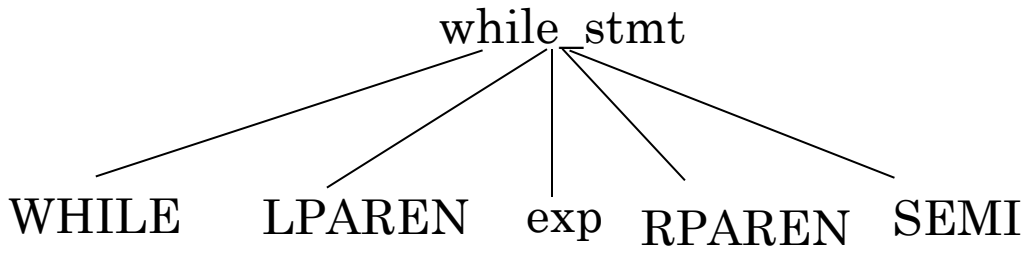
普通函数，含两个参数

主函数，引用了别的函数参与运算

测试结果：

```
Syntax tree:
  Normal function: fun
    Return type:int
    Params:
      Integer parameter: a
    Return:
      Op: +
        Id: a
        Const: 1
  Normal function: min
    Return type:int
    Params:
      Integer parameter: a
      Integer parameter: b
    Define:
      Id: temp
      Const: 0
    Return:
      Id: a
  Main function: main
    Return type:int
    Define:
      Id: x
      Const: 1
    Define:
      Id: a
    Assign to: a
      Call function:fun
        Id: x
    Return:
      Id: x
```

4. While语句实现



While语句的EBNF推导为：

statement→ while_stmt

|…

while_stmt → WHILE LPAREN exp RPAREN

使用递归下降分析。

3. 函数

编写测试文件：

```
1  int main()
2  begin
3      int x = 10;
4      while (x>0)
5          do x := x-1
6      end
7  end
```

while语句

测试结果：

```
Syntax tree:
  Main function: main
    Return type:int
    Define:
      Id: x
      Const: 10
    While
      Id: x
      Assign to: x
        Op: -
          Id: x
          Const: 1
```

谢谢！

陆逸凡

20191410901163