基于C0文法的编译器V2.0

10011603 2016302731 冯宇

**一.主要功能**

添加以下语法，实现函数的定义和调用，以及switch语句及读写语句

＜有返回值函数定义＞ ::= ＜声明头部＞‘(’＜参数＞|＜空＞‘)’ ‘{’＜复合语句＞‘}’

＜无返回值函数定义＞ ::= void＜标识符＞‘(’＜参数＞|＜空＞‘)’‘{’＜复合语句＞‘}’

＜参数＞ ::= ＜参数表＞

＜参数表＞ ::= ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}

＜情况语句＞ ::= switch ‘(’＜表达式＞‘)’ ‘{’＜情况表＞＜缺省＞‘}’

＜情况表＞ ::= ＜情况子语句＞{＜情况子语句＞}

＜情况子语句＞ ::= case＜常量＞:＜语句＞

＜缺省＞ ::= default:＜语句＞

＜读语句＞ ::= scanf ‘(’＜标识符＞{,＜标识符＞}‘)’

＜写语句＞ ::= printf ‘(’ ＜字符串＞,＜表达式＞ ‘)’| printf ‘(’＜字符串＞‘)’| printf ‘(’＜表达式＞‘)’

＜有返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞|<空>‘)’

＜无返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞|<空>‘)’

＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}

**二.代码实现**

// ＜有返回值函数定义＞ ::= ＜声明头部＞‘(’＜参数＞|＜空＞‘)’ ‘{’＜复合语句＞‘}’

void funcWithRetDef(){

defHead（&returnType, idName）读取函数返回类型并保存以函数的形式保存函数名至id\_table和infixTable

level++;等级增加，表示进入更深一层函数

参数读取；

complexState();对于函数中语句的读取处理；

popLocals();将本函数中的变量从idTable中pop;

level--；返回上一层函数，等级-1

}

// ＜无返回值函数定义＞ ::= void＜标识符＞‘(’＜参数＞|＜空＞‘)’‘{’＜复合语句＞‘}’

void funcWithoutRetDef(){

无返回类型和数，直接在本函数中将读取到的函数名以函数的形式保存至id\_table和infixTable

level++;等级增加，表示进入更深一层函数

参数读取；

complexState();对于函数中语句的读取处理；

popLocals();将本函数中的变量从idTable中pop;

level--；返回上一层函数，等级-1

}

// ＜参数＞ ::= ＜参数表＞

void parameter()

// ＜参数表＞ ::= ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}

void parameterTable(){

循环读取参数并记录

insertTable(params, paramType, token, 0, level, 0);

insertInfix("PARAM", " ", paramType == ints ? "int" : "char", token);

}

// ＜情况语句＞ ::= switch ‘(’＜表达式＞‘)’ ‘{’＜情况表＞＜缺省＞‘}’

void conditionState(){

endlable为switch语句结束的标志符

读取switch标志符

expression(switchElement)读取括号及其表达式，表达式的结果用switchElement存储

conditionTable(firstType, switchElement, endLabel);情况表函数

conditionDefault();缺省函数

}

// ＜情况表＞ ::= ＜情况子语句＞{＜情况子语句＞}

void conditionTable(type firstType, string switchElement, string endSwitchLabel)

// ＜情况子语句＞ ::= case＜常量＞:＜语句＞

void conditionBranch(type firstType, string switchElement, string endSwitchLabel){

读取case标志符

nextCaseLabel = labelStr + intToString(labelIndex);获得下一个语句的Lable

insertInfix("BNE", switchElement, caseElement, nextCaseLabel);当本语句case不符合条件，跳转到下一条case语句

statement();读取本case的语句

insertInfix("JMP", " ", " ", endSwitchLabel);进入本case语句后跳转到witch语句结束标志符

insertInfix("LABEL", " ", " ", nextCaseLabel);下一条case语句开始的Lable

}

// ＜缺省＞ ::= default:＜语句＞

void conditionDefault()

// ＜读语句＞ ::= scanf ‘(’＜标识符＞{,＜标识符＞}‘)’

void scanfState(){

利用循环读取标识符，需要注意的是，在读取标识符的时候有以下代码用于判断读取到的变量名是否已经被声明，没有则报错。

if ((idTable[idIndex].cls != vars && idTable[idIndex].cls != params ||idTable[idIndex].length != 0) {

error(IdKindNotMatch); // Identifier kind does not match

}

}

// ＜写语句＞ ::= printf ‘(’ ＜字符串＞,＜表达式＞ ‘)’| printf ‘(’＜字符串＞‘)’| printf ‘(’＜表达式＞‘)’

void printfState()

// ＜有返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞|<空>‘)’

void funcWithRetCall(string &infixString){

读取函数名，用idIndex = lookUp(token);从idTable中查找变量名序号

并借此判断声明变量类型是否为函数

if (idTable[idIndex].cls != funcs || idTable[idIndex].typ == voids)

valueParamTable(idIndex);读取参数

在infixtable中添加CALL语句调用函数

使用ASSIGN语句将返回值从#RET付给创建的临时变量

}

// ＜无返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞|<空>‘)’

void funcWithoutRetCall()

// ＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}

void valueParamTable(int idIndex){

根据函数声明时的参数个数，对相应的参数使用PUSH指令

}

**三.测试结果**

|  |  |
| --- | --- |
| Input code（scanf和printf） | 四元式 |
| int main(){  int ax,bx;  int i;  bx=0;  i=0;  scanf(ax);  printf(bx);  return;  } | int main()  int ax  int bx  int i  i = 0  bx = 0  scan ax  print bx  return |
| Input code(switch语句) | 四元式 |
| const int a=2;  const char c='Q';  int main(){  int x,y,i;  i=3;  switch(i){  case 1: i=i+1;  case 2: i=i\*2;  case 3: i=9;  default:i=1;  }  return ;  } | const int a = 2  const char c = Q  int main()  int x  int y  int i  i = 3  goto LABEL1, if i != 1  #t0 = i + 1  i = #t0  jmp LABEL0  LABEL1  goto LABEL2, if i != 2  #t1 = i \* 2  i = #t1  jmp LABEL0  LABEL2  goto LABEL3, if i != 3  i = 9  jmp LABEL0  LABEL3  i = 1  LABEL0  return |
| Input code（函数调用） | 四元式 |
| int compute(int x1,int x2)  {  int y1;  y1=x1+x2;  return (y1);  }  int main(){  int x1,x2;  x1=1;  x2=5;  compute(x1,x2);  return;  } | int compute()  param int x1  param int x2  int y1  #t0 = x1 + x2  y1 = #t0  return y1  int main()  int x1  int x2  x1 = 1  x2 = 5  push x1  push x2  call compute  #t1 = #RET  return |

**四.Debug以及C++知识点**

**1.程序出现报错的处理方式**

在程序对输入代码读取时，如果输入的代码中出现错误，那么这个时候会有以下函数用于报错并跳到“；”“）”“}”关键字以继续编译输入代码。

error(IdLost); // Identifier lost

symbol nexts[] = {semicolon, rbrace};

skipUntil(nexts);

*void* skipUntil(symbol *nexts*[]) {

while (!inNexts(nexts, sy)) {

insymbol();

}

skipFlag = true;

}

*bool* inNexts(symbol *nexts*[], symbol *tempSy*) {

for (*int* i = 0; i < sizeof(nexts) / sizeof(symbol); ++i) {

if (nexts[i] == tempSy) {

return true;

}

}

return false;

}

但原本程序是想要传递一个数组，然后在函数里求出数组长度，并进行处理，这样是会出错的，根本原因在于C/C++ 传递数组，虽然传递的是首地址地址，但是参数到了函数内就成了普通指针，不再是数组首地址了，所以试图在别的函数中得到传递数组的长度是行不通的。只能先计算好长度后再传过去，进行其他的运算。

修改后的代码如下：

error(IfSyLost); // If symbol lost

symbol nexts[] = { lparent, rparent, semicolon, rbrace };

int length = sizeof(nexts) / sizeof(symbol);

skipUntil(nexts, length);

void skipUntil(symbol nexts[],int length) {

while (!inNexts(nexts, sy, length)) {

insymbol();

}

skipFlag = true;

}

//next[]集中数据和传入的tempSy比较，有相同的返回TRUE

bool inNexts(symbol nexts[], symbol tempSy,int length) {

for (int i = 0; i < length; ++i) {

if (nexts[i] == tempSy) {

return true;

}

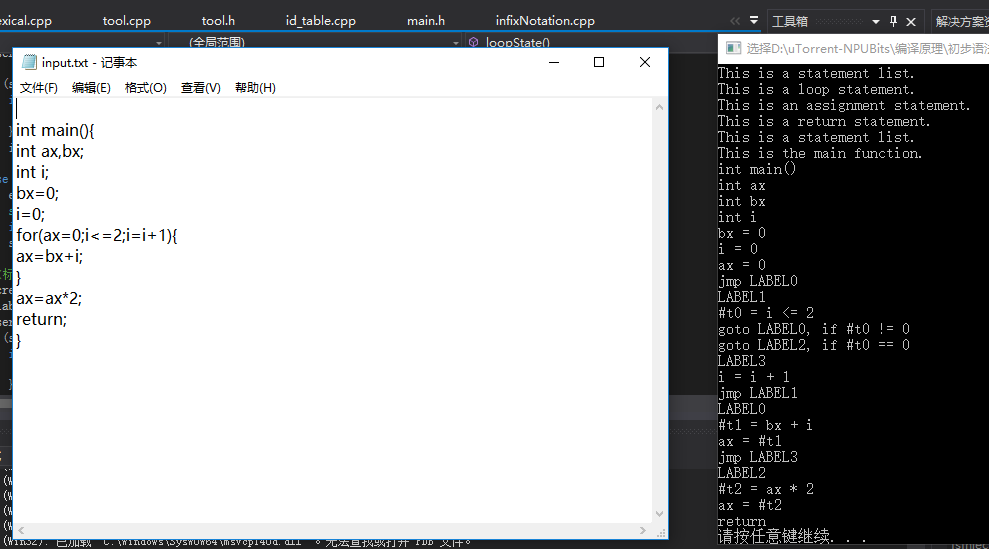
}

return false;

}

**2.原本的for循环错误**

在原本的for语句中，有一个地方的LABEL设置错误，按照原本的for循环执行，四元式如下：



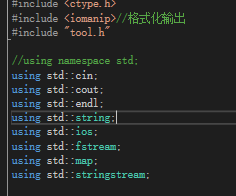
JUMP LABEL0需要修改，出现明显错误。需要删去该四元式

**3. strcpy()报错**

在VS2017中strcpy()函数是会报错的，在VS2017中使用了安全的函数strcpy\_s（），必须修改否则会一直报错，但是思路上是没错的

**4.避免plus和sub与默认的冲突**

plus和sub在使用std的namesapce情况下无法定义，会与默认的冲突。所以在头文件定义命名空间的时候采用的是：



而不是直接使用using namespace std;

避免和默认定义的冲突，能够将plus和sub定义为枚举量

**5.变量类型的转化**

在该程序中整数和String，以及char类型的转化不可避免，我们通过以下函数进行转化：

int strToInt(string str) {

stringstream ss(str);

int integer;

ss >> integer;

return integer;

}

char strToChar(string str) {

stringstream ss(str);

char character;

ss >> character;

return character;

}

string intToString(int num)

{

stringstream ss;

ss << num;

return ss.str();

}

string charToString(char c)

{

string str;

stringstream stream;

stream << c;

str = stream.str();

return str;

}