**SLR1分析程序实现**

**[总体思路]**

先读取action和goto两张表，然后创建若干个数组，分别作为状态栈，符号栈和各种属性栈（包括但不限于变量名，数据，代码，预留的标号等），创建一指针始终指向栈顶位置，另一指针指向输入串当前输入符号，在输入串消耗完毕前，每一次操作始终用当前状态和下一个输入符号查表，进行状态转移。

1. **action和goto表的存储**

由于输入符号为字符，因此想到用 map来存储



对于goto表，string部分为输入字符，int部分为要跳转的状态，对于action表，string部分为字符，pair部分分别为动作+移进动作下要转移的状态/归约动作下要使用的归约式，其中动作分为报错、移进、归约、接受。对于action表，没有操作的地方pair.first存储3代表出错，对于goto表，没有状态跳转则存储-1代表出错。

1. **文法的修改**

在SLR1分析程序构建的过程中，我们主要发现文法具有以下问题

1. +和\*运算符的优先级问题，原文法未考虑两运算符的实际意义下的优先级，仅为靠前有限，修改方案为：

**E->E+G|G**

**G->i|d|d[F]|d(F)|G\*G**

1. 语句末尾分号问题，在原文法中，单条语句不包含分号，而多条语句归约为语句群的时候可在语句之间增加分号，这导致简单语句最后一条语句末尾不应有分号，并且{}括起的语句块后需要有分号，这与实际编程习惯不符，修改方案为：

**V->S|VS**

**S->d=E;|if(B)S|if(B)SelseS|while(B)S|returnE;|{V}|d(F);**

1. **属性文法的构建（按归约式给出）**
2. **A->P** 无
3. **P->CV** 合并两部分代码即可 code[sp-1]=code[sp-1]+"|"+code[sp]
4. **C->e** 无
5. **C->CD；**合并两部分代码即可 code[sp-2]=code[sp-2]+"|"+code[sp-1]
6. **V->S** 无
7. **V->VS** 合并两部分代码即可 code[sp-1]=code[sp-1]+"|"+code[sp]
8. **D->Td** 调用符号表操作，d的名字登记为T的类型
9. **D->Td[I]** 调用符号表操作，d的名字登记为array，元素类型登记为T的类型，并存储维数维长等信息（存储在I的属性中）
10. **D->Td(C){CV}** 由于形参和实参已在符号表中登记（具体方法在第4部分中说明），并且代码部分V也已经生成完毕，因此将d登记为function，返回类型为T的类型，添加上调取形参的代码即可

code[sp-8]=prop[sp-7]+"proc|"+mcode+"|"+code[sp-1]+"|"+prop[sp-7]+"endp" （其中mcode部分为pop形参名字的代码）

1. **T->int** 将T的类型置为int
2. **T->float** 将T的类型置为float
3. **T->void** 将T的类型置为void
4. **I->i** 将I的维数置为1，第1维的维长置为i的数值（从实际意义来说，I是数值序列，为数组定义服务）
5. **I->I,i** 将I[0]的维数加一，且新的维长为i的数值
6. **S->d=E** 赋值语句，在E的代码上添加 d.name=E.place即可

code[sp-3]=code[sp-1]+"|"+prop[sp-3]+"="+place[sp-1]

1. **S->if(B)S** B中以自带tc和fc，分别为布尔表达式成立时跳转下标和不成立时跳转下标，分别置S的代码之前和之后即可

code[sp-4]=code[sp-2]+"|"+tc[sp-2]+":|"+code[sp]+"|"+fc[sp-2]+":"

1. **S->if(B)SelseS** 和（16）的情况基本相似，不同的是在S[1]的代码执行完毕后需要跳转到S[2]之后，这是一个新的地址标号

l=getnewlabel()

code[sp-6]=code[sp-4]+"|"+tc[sp-4]+":|"+code[sp-2]+"|jmp"+l+"|"+fc[sp-4]+":|"+code[sp]+"|"+l

1. **S->while(B)S**

l=getnewlabel()

code[sp-4]=l+":|"+code[sp-2]+"|"+tc[sp-2]+":|"+code[sp]+"|jmp"+l+"|"+fc[sp-2]+":"

1. **S->returnE** 在E的代码基础上将E.place压进栈即可

code[sp-2]=code[sp-2]+"|push "+place[sp-1]

1. **S->d(F);** 这里由于S是一个语句，不具有place属性，因此只调用函数的过程而不保存返回值，因此若函数的类型不为int，需要将栈中保存的返回值结果弹出 code[sp-4]=code[sp-1]+"|"+mcode+"|call "+prop[sp-4] （其中mcode为push形参名字代码）
2. **F->E** 无
3. **F->F,E** 将二者代码合并即可

code[sp-2]=code[sp-2]+"|"+code[sp];

place[sp-2]=place[sp-2]+"|"+place[sp];

1. **E->E+G** 创建一个新变量，将两个place相加即可

t=getnewvar()

code[sp-2]=code[sp-2]+"|"+code[sp]+"|"+t+"="+place[sp-2]+"+"+place[sp]

place[sp-2]=t;

1. **E->G** 无
2. **G->I** 新建一个变量，赋值为i的数值即可

t=getnewvar()

code[sp]=t+"="+prop[sp]

1. **G->d** G.place=d.name
2. **G->d[F]** 调用数组，首先通过查找符号表得到维数和维长信息，再通过F中的所有变量集合可结合维长算出偏移地址，具体代码见源程序case（27）
3. **G->d(F)** 与第（20）中情况基本相同，区别在于G是为复制服务的，因此如果d的类型是void不具有返回值，应当报错，并且调用d完毕后，应当新建一个变量名保存返回值
4. **G->G\*G** 与第（23）中情况基本相同，将+改为\*即可

t=getnewvar()

code[sp-2]+="|"+code[sp]+"|"+t+"="+place[sp-2]+"\*"+place[sp]

place[sp-2]=t

1. **B->ErE** 对于布尔表达式需要创建tc和fc标号并生成if语句

tc[sp-2]=getnewlabel()

fc[sp-2]=getnewlabel()

code[sp-2]=code[sp-2]+"|"+code[sp]+"|"+"if"+place[sp-2]+prop[sp-1]+place[sp]+"then jmp "+tc[sp-2]+" else jmp "+fc[sp-2]

1. **B->E** 和第（30）中情况基本相同，区别在于将ErE的表达式替换为E!=0

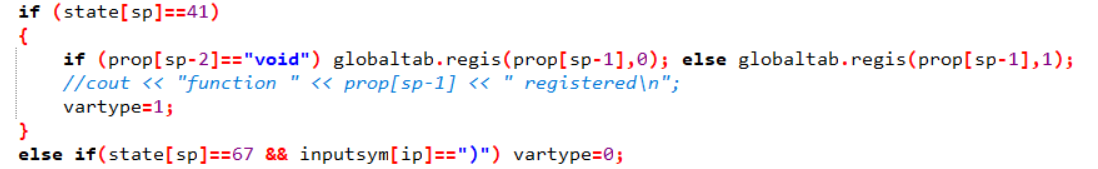
tc[sp]=getnewlabel()

fc[sp]=getnewlabel()

code[sp]=code[sp]+"|"+"if "+place[sp]+"!=0 then jmp "+tc[sp]+" else jmp "+fc[sp]

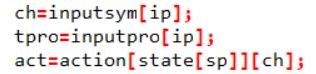
1. **登记符号表操作的具体实现**

登记符号表采用提前登记函数名称的方式实现分层，即开始定义某一函数d时，先登记d为function类型，此时所有的登记都会被视为在d之内，当函数声明结束，即出现归约式**D->Td(C){CV}**时，调用符号表的end（）功能，恢复到直接外侧。因此我们采取提前声明函数的方式，在每次移进动作之后，当到达项目**D->Td(🞄C){CV}**所在的项目集状态（41号状态）时，视为开始定义函数，从这时起所有定义变量登记符号表时均置为“形参”。当到达项目**D->Td(C){ 🞄CV}**，视为开始定义函数中的实参，从这时开始所有变量登记符号表时均置为“实参”。通过这种方式实现了归约出定义式时立刻登记，可节省存储空间，简化代码，并且，这两种状态也只可能通过函数声明到达。



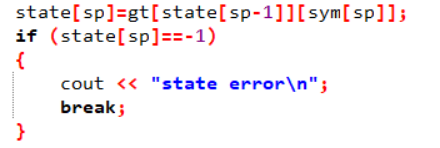
1. **SLR1分析过程的实现**

Sp指针指向状态栈顶，ip指向输入串当前位置，每一次操作，用最顶状态和当前输入符号查询action表，获得act（pair类型），第一项为操作类型，第二项为对应的附属信息（对于移进操作，为新的状态编号，对于归约操作，为所用归约式编号）



（其中inputsym为输入串，inputpro为变量名，数据或类型等基础属性，act为操作）

1. 若为接受项目遇到“#”，则输出“done”，跳转到中间代码输出printcode（）函数，程序结束
2. 若没有操作，则输入串的出现是出错，输出error，结束程序
3. 若为移进操作，则ip+1，sp+1，state[sp]=act.second（刚查找的新的状态）sym[sp]=ch（此次输入符号）
4. 若为归约操作，则进行两个步骤。先根据所用的归约式编号进行归约，主要任务在属性的计算（详见代码case（0）至case（31）的部分，分别对应32条归约式）归约之后，根据弹栈之后栈顶的状态和归约所得的非终结符查找goto表，得到新的栈顶状态



1. **实验结果**