## 实验四 LR(1)分析法

**一、实验目的：**

**构造LR(1)分析程序，利用它进行语法分析，判断给出的符号串是否为该文法识别的句子，了解LR（K）分析方法是严格的从左向右扫描，和自底向上的语法分析方法。**

**二、实验预习提示**

**1、使用LR(1)的优点：**

**(1)LR分析器能够构造来识别所有能用上下文无关文法写的程序设计语言的结构。**

**(2)LR分析方法是已知的最一般的无回溯移进-归约方法，它能够和其他移进-归约方法一样有效地实现。**

**(3)LR方法能分析的文法类是预测分析法能分析的文法类的真超集。**

**(4)LR分析器能及时察觉语法错误,快到自左向右扫描输入的最大可能。**

**为了使一个文法是LR的，只要保证当句柄出现在栈顶时，自左向右扫描的移进-归约分析器能够及时识别它便足够了。当句柄出现在栈顶时，LR分析器必须要扫描整个栈就可以知道这一点，栈顶的状态符号包含了所需要的一切信息。如果仅知道栈内的文法符号就能确定栈顶是什么句柄。LR分析表的转移函数本质上就是这样的有限自动机。不过，这个有限自动机不需要根据每步动作读栈，因为，如果这个识别句柄的有限自动机自底向上读栈中的文法符号的话，它达到的状态正是这时栈顶的状态符号所表示的状态，所以，LR分析器可以从栈顶的状态确定它需要从栈中了解的一切。**

**2、LR分析器由三个部分组成：**

**(1)总控程序，也可以称为驱动程序。对所有的LR分析器总控程序都是相同的。**

**(2)分析表或分析函数，不同的文法分析表将不同，同一个文法采用的LR分析器不同时，分析表将不同，分析表又可以分为动作表（ACTION）和状态转换（GOTO）表两个部分，它们都可用二维数组表示。**

**(3)分析栈，包括文法符号栈和相应的状态栈，它们均是先进后出栈。**

**分析器的动作就是由栈顶状态和当前输入符号所决定。**

**LR分析器结构：**

输入串XXX…#

输出

Xn

。

。

。

X1

#

n1

。

。

。

1

0

总控程序

ACTION

表

GOTO

表

**其中:SP为栈指针，S[i]为状态栈，X[i]为文法符号栈。状态转换表用GOTO[i，X]=j表示，规定当栈顶状态为i，遇到当前文法符号为X时应转向状态j，X为终结符或非终结符。**

**ACTION[i，a]规定了栈顶状态为i时遇到输入符号a应执行。动作有四种可能：**

**(1)移进：**

**action[i，a]= Sj：状态j移入到状态栈，把a移入到文法符号栈，其中i,j表示状态号。**

**(2)归约：**

**action[i，a]=rk：当在栈顶形成句柄时，则归约为相应的非终结符A，即文法中有A->B的产生式，若B的长度为R(即|B|=R)，则从状态栈和文法符号栈中自顶向下去掉R个符号，即栈指针SP减去R，并把A移入文法符号栈内，j=GOTO[i,A]移进状态栈，其中i为修改指针后的栈顶状态。**

**(3)接受acc:**

**当归约到文法符号栈中只剩文法的开始符号S时，并且输入符号串已结束即当前输入符是'#'，则为分析成功。**

**(4)报错:**

**当遇到状态栈顶为某一状态下出现不该遇到的文法符号时，则报错，说明输入端不是该文法能接受的符号串。**

**3、LR（1）分析法实验设计思想及算法**

否

否

是

是

否

是

0，#分别入状态栈和符号栈

置ip指向w#的第一个符号

令s是状态栈栈顶，

a是ip所指向的符号

action[s，a]=Ss’

action[s,a]=reduce A->β

把a和s’分别推入符号栈和状态栈；使ip前进到下一个字符

分别从栈顶弹出|β|个符号，令s’是当前栈顶状态，把a和goto[s’,A]先后推入栈中，输出产生式A->β

Action[A,a]=accept

结束

出错处理

**三、实验过程和指导：**

**（一）准备：**

**1.阅读课本有关章节，**

**2.考虑好设计方案；**

**3.设计出模块结构、测试数据，初步编制好程序。**

**（二）上课上机：**

**将源代码拷贝到机上调试，发现错误，再修改完善。**

**（三）程序要求：**

**程序输入/输出示例：**

**对下列文法，用LR（1）分析法对任意输入的符号串进行分析：**

**（1）E->E+T**

**（2）E->E—T**

**（3）T->T\*F**

**（4）T->T/F**

**（5）F->(E)**

**（6）F->i**

**输出的格式如下：**

**(1)LR（1）分析程序，编制人：姓名，学号，班级**

**(2)输入一以#结束的符号串(包括+—\*/（）i#)：在此位置输入符号串**

**(3)输出过程如下：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **步骤** | **状态栈** | **符号栈** | **剩余输入串** | **动作** |
| **1** | **0** | **#** | **i+i\*i#** | **移进** |

**(4)输入符号串为非法符号串(或者为合法符号串)**

**备注：(1)在“所用产生式”一列中如果对应有推导则写出所用产生式；如果为匹配终结符则写明匹配的终结符；如分析异常出错则写为“分析出错”；若成功结束则写为“分析成功”。**

**(2) 在此位置输入符号串为用户自行输入的符号串。**

**注意：1.表达式中允许使用运算符（+-\*/）、分割符（括号）、字符i，结束符#；**

**2.如果遇到错误的表达式，应输出错误提示信息（该信息越详细越好）；**

**3.对学有余力的同学，测试用的表达式事先放在文本文件中，一行存放一个表达式，同时以分号分割。同时将预期的输出结果写在另一个文本文件中，以便和输出进行对照；**

**（四）程序思路（仅供参考）：**

**模块结构：**

**（1）定义部分：定义常量、变量、数据结构。**

**（2）初始化：设立LR(1)分析表、初始化变量空间（包括堆栈、结构体、数组、临时变量等）；**

**（3）控制部分：从键盘输入一个表达式符号串；**

**（4）利用LR(1)分析算法进行表达式处理：根据LR(1)分析表对表达式符号串进行堆栈（或其他）操作，输出分析结果，如果遇到错误则显示错误信息。**

**（五）练习该实验的目的和思路：**

**程序相当复杂，需要利用到大量的编译原理，也用到了大量编程技巧和数据结构，通过这个练习可大大提高软件开发能力。**

**（六）为了能设计好程序，注意以下事情：**

**1.模块设计：将程序分成合理的多个模块（函数），每个模块做具体的同一事情。**

**2.写出（画出）设计方案：模块关系简图、流程图、全局变量、函数接口等。**

**3.编程时注意编程风格：空行的使用、注释的使用、缩进的使用等。**

**四、上交：**

**1.程序源代码（上交软盘）；**

**2.已经测试通过的测试数据；**

**3.实验报告：**

**内容如下：**

**实验名称**

**实验目的和要求**

**（一）实验内容**

**（1）功能描述：该程序具有什么功能？**

**（2）程序结构描述：函数调用格式、参数含义、返回值描述、函数功能；函数之间的调用关系图。**

**（3）程序总体执行流程图**

**（二）实验过程记录：出错次数、出错严重程度、解决办法摘要。**

**（三）实验总结：你在编程过程中花时多少？多少时间在纸上设计？多少时间上机输入和调试？多少时间在思考问题？遇到了哪些难题？你是怎么克服的？你对你的程序的评价？你的收获有哪些？**

**五、参考源代码：**

**#include<stdio.h>**

**#include<math.h>**

**#define max 100**

**char ex[max]; /\*存储后缀表达式\*/**

**void trans(){ /\*将算术表达式转化为后缀表达式\*/**

**char str[max]; /\*存储原算术表达式\*/**

**char stack[max]; /\*作为栈使用\*/**

**char ch;**

**int sum,i,j,t,top=0;**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("\*输入一个求值的表达式，以#结束。\*\n");**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("算数表达式：");**

**i=0; /\*获取用户输入的表达式\*/**

**do{**

**i++;**

**scanf("%c",&str[i]);**

**}while(str[i]!='#' && i!=max);**

**sum=i;**

**t=1;i=1;**

**ch=str[i];i++;**

**while(ch!='#'){**

**switch(ch){**

**case '(': /\*判定为左括号\*/**

**top++;stack[top]=ch;**

**break;**

**case ')': /\*判定为右括号\*/**

**while(stack[top]!='('){**

**ex[t]=stack[top];top--;t++;**

**}**

**top--;**

**break;**

**case '+': /\*判定为加减号\*/**

**case '-':**

**while(top!=0&&stack[top]!='('){**

**ex[t]=stack[top];top--;t++;**

**}**

**top++;stack[top]=ch;**

**break;**

**case '\*': /\*判定为乘除号\*/**

**case '/':**

**while(stack[top]=='\*'||stack[top]=='/'){**

**ex[t]=stack[top];top--;t++;**

**}**

**top++;stack[top]=ch;**

**break;**

**case ' ':break;**

**default:while(ch>='0'&&ch<='9'){ /\*判定为数字\*/**

**ex[t]=ch;t++;**

**ch=str[i];i++;**

**}**

**i--;**

**ex[t]='#';t++; /\*用#分隔各个操作数\*/**

**}**

**ch=str[i];i++;**

**}**

**while(top!=0){**

**ex[t]=stack[top];t++;top--;**

**}**

**ex[t]='#';**

**printf("\n\t原来表达式：");**

**for(j=1;j<sum;j++)**

**printf("%c",str[j]);**

**printf("\n\t后缀表达式：",ex);**

**for(j=1;j<t;j++)**

**printf("%c",ex[j]);**

**}**

**void compvalue(){ /\*计算后缀表达式的值\*/**

**float stack[max],d; /\*作为栈使用\*/**

**char ch;**

**int t=1,top=0; /\*t为ex下标，top为stack下标\*/**

**ch=ex[t];t++;**

**while(ch!='#'){**

**switch(ch){**

**case '+':**

**stack[top-1]=stack[top-1]+stack[top];**

**top--;**

**break;**

**case '-':**

**stack[top-1]=stack[top-1]-stack[top];**

**top--;**

**break;**

**case '\*':**

**stack[top-1]=stack[top-1]\*stack[top];**

**top--;**

**break;**

**case '/':**

**if(stack[top]!=0)**

**stack[top-1]=stack[top-1]/stack[top];**

**else{**

**printf("\n\t除零错误!\n");**

**exit(0); /\*异常退出\*/**

**}**

**top--;**

**break;**

**default:**

**d=0;**

**while(ch>='0'&&ch<='9'){**

**d=10\*d+ch-'0'; /\*将数字字符转化为对应的数值\*/**

**ch=ex[t];t++;**

**}**

**top++;**

**stack[top]=d;**

**}**

**ch=ex[t];t++;**

**}**

**printf("\n\t计算结果:%g\n",stack[top]);**

**}**

**main(){**

**trans();**

**compvalue();**

**}**

**模块设计**

N

调用

返回

缓冲区

扫描一个字符

缓冲区

主函数main()

N

输入文件名，判断能否打开文件

Y

缓冲区中是否还有字符

Y

结束

取单词

扫描一个字符

调用

返回

输出