**实验名称：用yacc实现语法分析**

**实验目的和要求**：

掌握移进-归约技术语法分析技术，利用语法生成工具yacc/bison实现语法分析器的构造。

**实验内容和步骤：**

1. **实验内容**

利用语法分析器生成工具Yacc/Bison编写一个语法分析程序，与词法分析器结合，能够根据语言的上下文无关文法，识别输入的单词序列是否文法的句子。

1. **实验步骤**
2. **关于yacc**

在进一步阐述以前，让我们复习一下什么是语法。在上一节中，我们看到 Lex 从输入序列中识别标记。 如果你在查看标记序列，你可能想在这一序列出现时执行某一动作。 这种情况下有效序列的规范称为语法。Yacc 语法文件包括这一语法规范。 它还包含了序列匹配时你想要做的事。为了更加说清这一概念，让我们以英语为例。 这一套标记可能是：名词, 动词, 形容词等等。为了使用这些标记造一个语法正确的句子，你的结构必须符合一定的规则。 一个简单的句子可能是名词+动词或者名词+动词+名词。(如 I care. See spot run.)。所以在我们这里，标记本身来自语言（Lex），并且标记序列允许用 Yacc 来指定这些标记(标记序列也叫语法)。

用 Yacc 来创建一个编译器包括四个步骤：

1. 通过在语法文件上运行 Yacc 生成一个解析器。
2. 说明语法：
   * 编写一个 .y 的语法文件（同时说明 C 在这里要进行的动作）。
   * 编写一个词法分析器来处理输入并将标记传递给解析器。 这可以使用 Lex 来完成。
   * 编写一个函数，通过调用 yyparse() 来开始解析。
   * 编写错误处理例程（如 yyerror()）。
3. 编译 Yacc 生成的代码以及其他相关的源文件。
4. 将目标文件链接到适当的可执行解析器库。
5. **编程思路**
6. 我先简单地编写了一个lex文件，来识别单词，有了上一次实验的基础，很快就完成了这件事情。
7. 编写yacc文件，这个和编译原理语法规则很像，写出要识别的语法即可。
8. 用E来代表空
9. **实验过程记录**
10. **源代码：**

lex文件：1.l

%{

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include "1.tab.h"

%}

letter [A-Za-z]

digit [0-9]

number {digit}+

delim [ \t\n]+

operation "-"|"+"|"\*"|"/"|"{"|"}"|"("|")"|"="|";"|"<"|">"|"["|"]"|"&"|"^"|"|"

id ({letter}|[\_])({letter}|{digit}|[\_])\*

%%

"main" {return MAIN;}

"if" {return IF;}

"else" {return ELSE;}

"for" {return FOR;}

"while" {return WHILE;}

"==" {return EQ;}

">=" {return GE;}

"<=" {return LE;}

"!=" {return NE;}

{operation} {return \*yytext;}

{id} {return ID;}

{number} {return NUM;}

{delim} {}

%%

int yywrap()

{

return 1;

}

Yacc文件 1.y

%{

#include <ctype.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

void yyerror(char \*s);

%}

%start program

%token ID NUM LE GE EQ NE MAIN IF FOR ELSE WHILE

%%

program: MAIN '(' ')' '{' block '}' {printf("program->main() {block}\n");}

;

block: block expression ';' {printf("block->block expression;\n");}

|block if {printf("block->block if\n");}

|block for {printf("block->block for\n");}

|block while {printf("block->block while\n");}

| {printf("block->E\n");}

;

expression: var '=' expression {printf("expression->var=expression\n");}

|simple\_expression {printf("expression->simple-expression\n");}

| {printf("expression->E\n");}

;

var: ID {printf("var->ID\n");}

|ID '[' expression ']' {printf("var->ID[expression]\n");}

;

simple\_expression:add\_expression operation add\_expression {printf("simple-expression->add-expression operation add-expression\n");}

|add\_expression {printf("simple-expression->add-expression\n");}

;

operation: LE {printf("operation-><=\n");}

| '<' {printf("operation-> <\n");}

|'>' {printf("operation-> >\n");}

|GE {printf("operation-> >=\n");}

|EQ {printf("operation-> ==\n");}

|NE {printf("operation-> !=\n");}

;

add\_expression:add\_expression calculate term {printf("add-expression->add-expression addop term\n");}

|term {printf("add-expression->term\n");}

;

calculate:'+' {printf("calculate->'+'\n");}

|'-' {printf("calculate->'-'\n");}

;

term:term mulop factor {printf("term->term mulop factor\n");}

|factor {printf("term->factor\n");}

;

mulop: '\*' {printf("mulop->'\*'\n");}

|'/' {printf("mulop->'/'\n");}

;

factor: '(' expression ')' {printf("factor->'('expression')'\n");}

|NUM {printf("factor->NUM\n");}

|var {printf("factor->var\n");}

;

for: FOR '(' expression ';' expression ';' expression ')' '{' block '}' {printf("for->FOR(expression;expression;expression)\n");}

;

if: IF '(' expression ')' '{' block '}' {printf("if->IF(expression){block}\n");}

| IF '(' expression ')' '{' block '}' else {printf("if->IF(expression){block} else\n");}

;

else: ELSE IF '(' expression ')' '{' block '}' else {printf("else->ELSE IF(expression){block} else\n");}

|ELSE '{' block '}' {printf("else->ELSE{block}\n");}

;

While: WHILE '(' expression ')' '{' block '}' {printf("while->WHILE(expression){block}\n");}

;

%%

int main() {

extern FILE \*yyin;

yyin = fopen("example.txt","r");

yyparse();

return 0;

}

void yyerror(char \*s){

printf("%s\n",s);

}

1. **测试数据**

Example.txt

main() {

a = 1;

if(a==1) {

b = 3;

c = 5;

}

for(i=1;i<2;) {

t=5;

}

While(c!=3){

c = c + 3;

}

}

1. **编译环境**

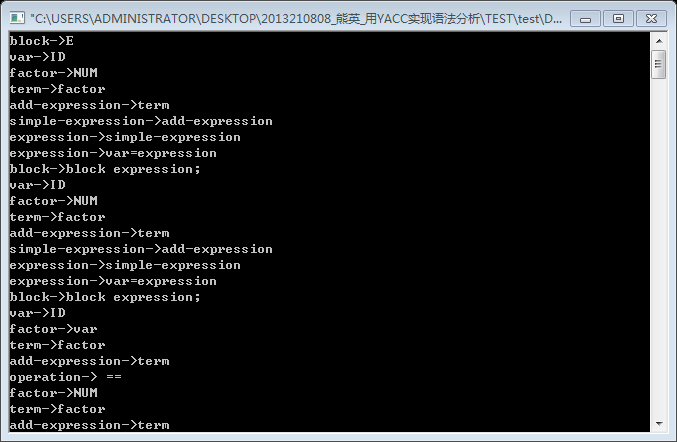
Window下的visual c++

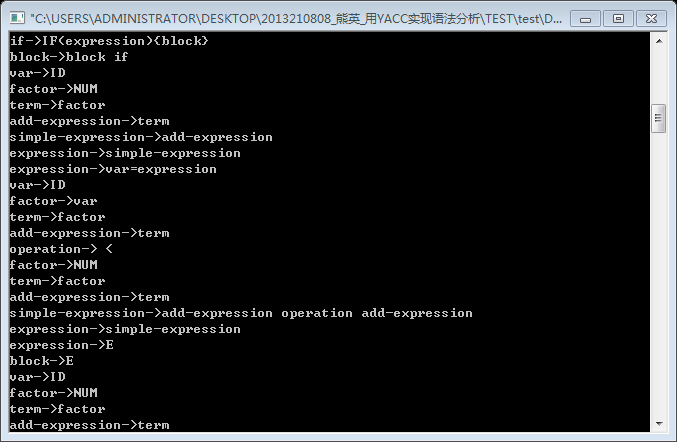
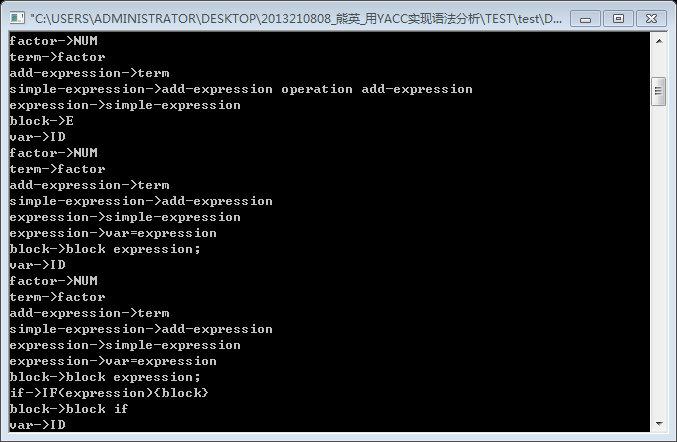
首先运用命令bison –d 1.y编译yacc文件

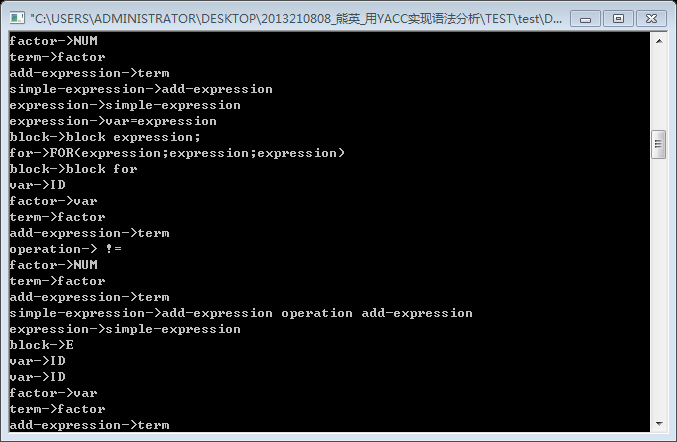
然后运用命令 flex 1.l 编译lex文件

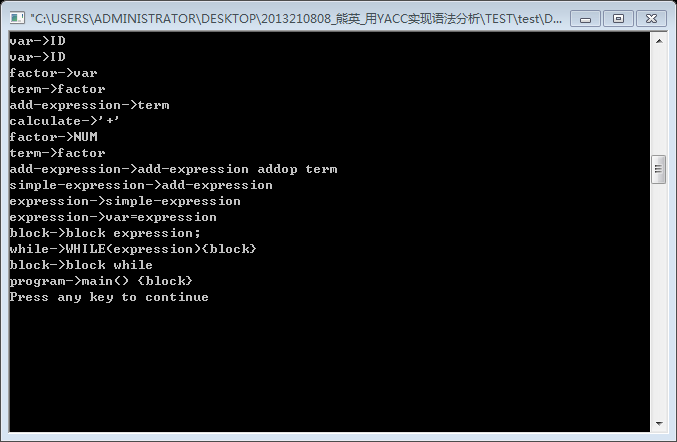
建一个工程，添加进去文件编译就行

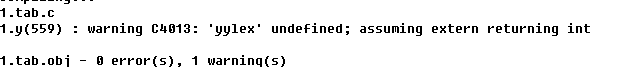
1. **结果**









1. 能识别的语法
2. 赋值语句
3. 布尔表达式
4. If else语句
5. For循环
6. While循环
7. **遇到的问题**
8. 理论总是很简单，但是实践起来还是遇到了很多很多困难，一直有很多移进-归约和归约-归约冲突解决不了，后来和同学讨论了很久才找到错误。这也是遇到的最大的困难。
9. 就是lex和yacc结合在一起，总是提醒没有声明关键字，然后才知道是自己的lex里面没有加yacc产生的c语言头文件。
10. 就是两个文件只能有一个主函数，虽然这是显而易见的问题，但是做的时候还是因为这个错误花费了很久的时间去理解。
11. **实验总结**
12. 实验真的花费了很长时间来做，最后都几乎要放弃了，主要是yacc语法还是比较广的，自己想都尽量用到，后来发现时间不允许，所以对程序进行了缩减，只是用了基础的东西，但是做出来还是特别欣慰。
13. 用yacc是一个很神奇的东西，由于有lex的基础，只要一心一意来做yacc就好，而且用起来很方便。很难想象yacc如此强大的功能。
14. 在学习yacc的过程中也发现yacc的用途之广，可以用作很多不同的地方，用来计算或者别的，而且速度也提高很多。
15. 实验中遇到了很多问题，在学yacc的途中就有很多不理解，后来看了很多资料然后和同学一起讨论，在理解。但是还是有问题没有解决，比如

一直有警告，但是仍然解决不了，所以最后也就没管了。