实验四： 语义分析与中间代码生成实验

一、实验目的

通过上机实习,加深对语法制导翻译原理的理解,掌握将语法分析所识别的语法成分变换为中间代码的语义翻译方法.

二、实验内容

采用递归下降语法制导翻译法,对算术表达式、赋值语句进行语义分析并生成四元式序列。

三实验要求

1. 实验要求
2. 文法：
3. program→begin block end

block→ stmt| stmt block

stmt→id:=expr;

*expr* → *expr* **+** *term*

| *expr* **-** *term*

| *term*

*term* → *term* **\*** *factor*

| *term* **/** *factor*

| *factor*

*factor* →  **( *e****xpr* **)** | **id**| **num**

1. 实现一遍扫描翻译

3、实验的输入和输出：

输入是语法分析提供的正确的单词串，输出为三地址指令形式的四元式序列。

例如：对于语句串

begin a:=2+3\*4;x:=(a+b)/c end#

输出的三地址指令如下：

t1=3\*4

t2=2+t1

a=t2

t3=a+b

t4=t3/c

x=t4

四、实验步骤

算法思想

1设置语义过程

emit(char \*result,char \*arg1,char \*op,char \*ag2)

该函数功能是生成一个三地址语句送到四元式表中。

四元式表的结构如下：

struct {char result[8];

char ag1[8];

char op[8];

char ag2[8];

}quad[20];

(2) char \*newtemp()

该函数回送一个新的临时变量名，临时变量名产生的顺序为T1,T2,….

Char \*newtemp(void)

{

char \*p;

char m[8];

p=(char \*)malloc(8);

k++;

itoa(k,m,10);

strcpy(p+1,m);

p[0]=’t’;

return(p);

}

2.主程序示意图如图c.10所示。



3.函数lrparser在原来语法分析的基础上插入相应的语义动作：将输入串翻译成四元式序列。在实验中我们只对表达式、赋值语句进行翻译。

四、实验过程原始记录（数据、图表、计算等）

实验的输入和输出：

输入是语法分析提供的正确的单词串，输出为三地址指令形式的四元式序列。

例如：对于语句串

begin a:=2+3\*4;x:=(a+b)/c end#

输出的三地址指令如下：

t1=3\*4

t2=2+t1

a=t2

t3=a+b

t4=t3/c

x=t4

五、实验报告要求

详细说明你的程序的设计思路和实现过程。

附录：

**#include "stdio.h"**

**#include "string.h"**

**#include "stdlib.h"**

**char prog[100],token[8],ch;**

**char \*rwtab[6]={"begin","if","then","while","do","end"};**

**int syn,p,m,n,sum,q;**

**int kk;**

**struct { char result1[8];**

**char ag11[8];**

**char op1[8];**

**char ag21[8];**

**} quad[20];**

**char \*factor();**

**char \*expression();**

**int yucu();**

**char \*term();**

**int statement();**

**int lrparser();**

**char \*newtemp();**

**scaner();**

**emit(char \*result,char \*ag1,char \*op,char \*ag2);**

**main()**

**{ int j;**

**q=p=kk=0;**

**printf("\nplease input a string (end with '#'): ");**

**do**

**{ scanf("%c",&ch);**

**prog[p++]=ch;**

**}while(ch!='#');**

**p=0;**

**scaner();**

**lrparser();**

**if(q>19)printf(" to long sentense!\n");**

**elsefor(j=0;j<q;j++)printf("%s=%s%s%s\n\n",quad[j].result1,quad[j].ag11,**

**quad[j].op1,quad[j].ag21);**

**//getch();**

**}**

**int lrparser()**

**{ int schain=0;**

**kk=0;**

**if (syn==1)**

**{ scaner(); //读下一个单词符号**

**schain=yucu(); //调用语句串分析函数进行分析**

**if(syn==6)**

**{ scaner(); //读下一个单词符号**

**if((syn==0)&&(kk==0)) printf("Success!\n"); //输出（“success”）**

**}**

**else { if(kk!=1)printf("short of 'end' !\n"); //输出 ‘缺end’ 错误**

**kk=1;**

**// getch();**

**exit(0);**

**}**

**}**

**else { printf("short of 'begin' !\n"); //输出’begin’错误**

**kk=1;**

**// getch();**

**exit(0);**

**}**

**return (schain);**

**}**

**int yucu()**

**{ int schain=0;**

**schain=statement(); //调用语句分析函数进行分析**

**while(syn==26)**

**{ scaner(); //读下一个单词符号**

**schain=statement(); //调用语句分析函数进行分析**

**}**

**return (schain);**

**}**

**int statement()**

**{ char tt[8],eplace[8];**

**int schain=0;**

**if (syn==10)**

**{ strcpy(tt,token);**

**scaner();**

**if(syn==18)**

**{ scaner(); //读下一个单词符号**

**strcpy(eplace,expression());**

**emit(tt,eplace,"","");**

**schain=0;**

**}**

**else { printf("short of sign ':=' !\n"); //输出’缺少赋值号’的错误**

**kk=1;**

**// getch();**

**exit(0);**

**}**

**return (schain);**

**}**

**}**

**char \*expression()**

**{ char \*tp,\*ep2,\*eplace,\*tt;**

**tp=(char \*)malloc(12); //分配空间**

**ep2=(char \*)malloc(12);**

**eplace=(char \*)malloc(12);**

**tt=(char \*)malloc(12);**

**strcpy(eplace,term()); //调用term分析产生表达式计算的第一项eplace**

**while((syn==13)||(syn==14))**

**{ if (syn==13)strcpy(tt,"+"); //操作符 tt= ‘+’或者‘—’**

**else strcpy(tt,"-");**

**scaner(); //读下一个单词符号**

**strcpy(ep2,term()); //调用term分析产生表达式计算的第二项ep2**

**strcpy(tp,newtemp()); //调用newtemp产生临时变量tp存储计算结果**

**emit(tp,eplace,tt,ep2); //生成四元式送入四元式表**

**strcpy(eplace,tp);**

**}**

**return (eplace);**

**}**

**char \*term() //仿照函数expression编写**

**{ char \*tp,\*ep2,\*eplace,\*tt;**

**tp=(char \*)malloc(12);**

**ep2=(char \*)malloc(12);**

**eplace=(char \*)malloc(12);**

**tt=(char \*)malloc(12);**

**strcpy(eplace,factor());**

**while((syn==15)||(syn==16))**

**{ if (syn==15)strcpy(tt,"\*");**

**else strcpy(tt,"/");**

**scaner(); //读下一个单词符号**

**strcpy(ep2,factor());**

**strcpy(tp,newtemp());**

**emit(tp,eplace,tt,ep2);**

**strcpy(eplace,tp);**

**}**

**return (eplace);**

**}**

**char \*factor()**

**{ char \*fplace;**

**fplace=(char \*)malloc(12);**

**strcpy(fplace,"");**

**if(syn==10)**

**{ strcpy(fplace,token);**

**scaner(); //读下一个单词符号**

**}**

**else if(syn==11)**

**{ itoa(sum,fplace,10);**

**scaner(); //读下一个单词符号**

**}**

**else if(syn==27)**

**{ scaner(); //读下一个单词符号**

**fplace=expression(); //调用expression分析返回表达式的值**

**if(syn==28) scaner(); //读下一个单词符号**

**else { printf("error on ')' !\n");**

**kk=1;**

**// getch();**

**exit(0);**

**}**

**}**

**else { printf("error on '(' !\n");**

**kk=1;**

**// getch();**

**exit(0);**

**}**

**return (fplace);**

**}**

**char \*newtemp()**

**{ char \*p;**

**char m[8];**

**p=(char \*)malloc(8);**

**kk++;**

**itoa(kk,m,10);**

**strcpy(p+1,m);**

**p[0]='t';**

**return(p);**

**}**

**scaner()**

**{ sum=0;**

**for(m=0;m<8;m++)token[m++]=NULL;**

**m=0;**

**ch=prog[p++];**

**while(ch==' ')ch=prog[p++];**

**if(((ch<='z')&&(ch>='a'))||((ch<='Z')&&(ch>='A')))**

**{ while(((ch<='z')&&(ch>='a'))||((ch<='Z')&&(ch>='A'))||((ch>='0')&&(ch<='9')))**

**{token[m++]=ch;**

**ch=prog[p++];**

**}**

**p--;**

**syn=10;**

**token[m++]='\0';**

**for(n=0;n<6;n++)**

**if(strcmp(token,rwtab[n])==0)**

**{ syn=n+1;**

**break;**

**}**

**}**

**else if((ch>='0')&&(ch<='9'))**

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

**{ sum=sum\*10+ch-'0';**

**ch=prog[p++];**

**}**

**p--;**

**syn=11;**

**}**

**else switch(ch)**

**{ case '<':m=0;**

**ch=prog[p++];**

**if(ch=='>')**

**{ syn=21;**

**}**

**else if(ch=='=')**

**{ syn=22;**

**}**

**else**

**{ syn=20;**

**p--;**

**}**

**break;**

**case '>':m=0;**

**ch=prog[p++];**

**if(ch=='=')**

**{ syn=24;**

**}**

**else**

**{ syn=23;**

**p--;**

**}**

**break;**

**case ':':m=0;**

**ch=prog[p++];**

**if(ch=='=')**

**{ syn=18;**

**}**

**else**

**{ syn=17;**

**p--;**

**}**

**break;**

**case '+': syn=13; break;**

**case '-': syn=14; break;**

**case '\*': syn=15;break;**

**case '/': syn=16;break;**

**case '(': syn=27;break;**

**case ')': syn=28;break;**

**case '=': syn=25;break;**

**case ';': syn=26;break;**

**case '#': syn=0;break;**

**default: syn=-1;break;**

**}**

**}**

**emit(char \*result,char \*ag1,char \*op,char \*ag2)**

**{**

**strcpy(quad[q].result1,result);**

**strcpy(quad[q].ag11,ag1);**

**strcpy(quad[q].op1,op);**

**strcpy(quad[q].ag21,ag2);**

**q++;**

**}**