In this final project you are will write a YACC program that translates PL/0 programs into a language called GOTOS. In particular we will use a slightly simplified version of the PL/0 grammar that was handed in the previous homework.

**Grammar**

Tokens: relation = { <,>,<>,<=>,<=,=}. ident: any sequence of letters and digits that starts with a letter. number: any sequence of digits (interpreted as type integer). reserved words **begin**, **call**, **do**, **end**, **if**, **procedure**, **then**, **var**, **while**, **writeln**. Punctuation marks . , ; := = + - * / ( )

```
<program>    ->  <block> .
[block]      ->  <const_decl> <vars_decl> <procs_decl> <statement>
<const_decl> ->  const <seqass> ;
<seqass>     ->  ident = number
                ident = number , <seqass>
<var_decl>   ->  var <seqident> ;
<seqident>   ->  ident
              ident , seqident
<proc_decl>  ->  <proc_decl> procedure ident ; <block> ;
<statement>  ->  ident ::= <expression>
                call ident
                begin <seqstmt> end
                while <condition> do <statement>
                if <condition> then <statement>
                writeln ( <expression> )
                <seqstmt>    ->  <statement>
                <statement> ; <seqstmt>
<condition>  ->  <expression> relation <expression>
<expression> ->  <expression> + <expression>
              <expression> - <expression>
              <expression> * <expression>
              <expression> / <expression>
              ident
              number
              ( <expression> )
```

In particular the changes from the previous grammar of homework #4 are as follows: (1) the handling of algebraic expressions is greatly simplified — note that this will require the
specification of precedence rules within yacc. (2) the unary operations and conditions of PL/0 have been removed, and (3) a new statement writeln has been added that prints in the stdout the value of an expression. You are supposed to extend the yacc/lex programs you submitted in homework #4 in order to complete the current project. Most importantly you are required to keep all the functionality that was achieved in the previous project (in particular your error-correction functions and the symbol-table handling).

The goal of this project is to translate any PL/0 program that adheres to the above definition to a program in the following language called “GOTOS”:

- **label** is a token that is the letter “L” followed by any sequence of digits.
- **variable** is a token that is the letter “V” or “T” followed by any sequence of digits.
- **number** is a token that corresponds to any sequence of digits.
- **op** is a token that is one of the operations $+,-,*,/$.
- **relation** is a token defined as in the case of PL/0 above.

Every statement in the language can be in one of the following forms:

- `goto label`
- `variable = variable`
- `variable = number`
- `variable = variable op variable`
- `if variable relation variable then goto label`
- `push (" label ")`
- `goto pop()`

**Rules for the GOTOS Language.**

- Each statement may be preceded by the token string “label :”
- Each statement must be terminated by ;
- A sequence of statements may be grouped within { and }
- this structure is called a block of statements.
- A block of statements will start with a sequence of declarations of the form `int variable` for all the variables that are used within the block. Blocks of statements maybe nested. The scope of a variable follows the usual rules with respect to blocks.
- Finally you may include a `printf` statement within a GOTOS program (following standard C semantics) as well as use comments as used in the C programming language.

Attributes should be defined inside YACC to handle the translation as shown below. In your semantic actions, you are supposed to used the standard functions of `string.h` that will assist you with basic string operations like `strcat` and `strcpy`.

typedef struct
{
    char translation[MAX_TRANSL];
    int variable;
    char condition\_handle[MAX_TRANSL];
} myattribute;

%union{
Note that every non-terminal will have three attributes: translation that will correspond to a string that carries the translation of the language construct that corresponds to the non-terminal; variable that will carry the variable number that holds the value that corresponds to the language construct (only meaningful for expressions) and finally condition_handle that will be useful for the condition non-terminal (note that only the translation attribute will be useful for all non-terminals; you are welcome to refine this by defining several different types of attributes in your yacc code — but you are not required to do so).

Example

In order to understand the way translation should work an example is provided below.

VAR n,f;
PROCEDURE go;
CONST a=0;
VAR n;
BEGIN n;
  a:=5;
  IF a>=3 THEN BEGIN a:=a-1; END END;
BEGIN f := 1;
  n := 0;
  WHILE n<=5 DO BEGIN n:=n+1; f:=f*n; END;
  CALL go;
  WRITELN(f);
END.

Note that regarding the two “kinds” of variables employed in the GOTOS translation of a PL/0 program, the ones that start with T should refer to identifiers installed in the symbol table (and in particular T0 should refer to the first constant or variable installed in the symbol
table, etc.) while the ones that start with V should be auxiliary variables that will be used to break expressions into the basic “1/2-operand restricted” assignments of the GOTOS language. These principles are reflected in the translation of the above program shown below:

```plaintext
int T0;
int T1;
int V0,V1,V2,V3,V4,V5,V6,V7,V8,V9,V10;
L10004:
    { /* Begin Procedure go */
    int T3 = 0;
    int T4;
    int V0,V1,V2,V3,V4,V5;
    V0 = 5;
    T3 = V0;
    /* Begin If Statement */
    V1 = T3;
    V2 = 3;
goto L10002;
L10001:
    V3 = T3;
    V4 = 1;
    V5 = V3-V4;
    T3 = V5;
goto L10003;
L10002:
    if (V1 >= V2) goto L10001;
L10003: /* End If Statement */
goto pop();
} /* End Procedure go */
V0 = 1;
T1 = V0;
V1 = 0;
T0 = V1;
L10005: /* Begin While Statement */
V2 = T0;
V3 = 5;
goto L10007;
L10006:
    V4 = T0;
    V5 = 1;
    V6 = V4+V5;
    T0 = V6;
    V7 = T1;
    V8 = T0;
    V9 = V7*V8;
    T1 = V9;
goto L10005;
L10007:
    if (V2 <= V3) goto L10006; /* End While Statement */
push("L10008");
goto L10004; /* calling procedure go */
L10008:
```
V10 = T1;
printf("%d\n", V10);

Try to read carefully and analyze the above file. The connection to the PL/0 program will be clear. Use it as a guide to construct your translation. Note that the GOTOS language is almost a subset of C and by adding a main function around your translation plus providing declarations necessary to implement the stack commands push and pop, your GOTOS program can be compiled by any C-compiler. A smaller example is below:

const a = 4;
var n, f;
beg
  f := (a * 3) + 2;
  if f <= 5 then
    begin
      n := f + 1;
    end;
end;

int T0 = 4;
int T1;
int T2;
int V0, V1, V2, V3, V4, V5, V6, V7, V8, V9;
V0 = T0;
V1 = 3;
V2 = V0 * V1;
V3 = 2;
V4 = V2 + V3;
T2 = V4;
/* Begin If Statement */
V5 = T2;
V6 = 5;
goto L10002;
L10001:
V7 = T2;
V8 = 1;
V9 = V7 + V8;
T1 = V9;
goto L10003;
L10002:
if (V5 <= V6) goto L10001;
L10003: /* End If Statement */

What to Submit

Submit (i) a description of your code (ii) the Yacc and Lex code (iii) the output of your program in the three sample files to be found in the web-site of the class (Use of “PRINTSCREEN” highly recommended).

The whole assignment should be prepared as a single file using MS Word or other word-processing / type-setting software package of your liking. The first page should contain only: (i) your name, (ii) the class number “CSE 244”, (iii) the semester “Fall 2003”, (iv) the homework number “Homework #5 — FINAL PROJECT”.

What to Submit