The purpose of this lecture is to explain how semantic information is obtained and produced using the lex and yacc tools. To that end, this lecture contains a simple expression calculator. You can find the calculator here and in folder ~tt/cs5023/lectures/L003.

| **File** | **Description** |
|---|---|
| mkcalc | Creates program calc from source files.<br>Usage:<br>**$** mkcalc |
| makecalc | A makefile that directs the make utility to create the executable file calc from source files.<br>Usage:<br>**$** make -f makecalc |
| calc.exp | Contains function main that initiates program execution |
| calclex.l | File calclex.l contains the specification for the scanner or *lexer*. To create a relocatable object, file calclex.l must be first translated by linux utility lex and then compiled using the C++ compiler. |
| calclex.h | Contains the interface to the lexer. |
| calcpar.y | File calcpar.y contains the specification for the parser or syntax analyzer. To create a relocatable object, file calcpar.y must first be translated by linux utility yacc and then compiled using the C++ compiler. |
| calcpar.h | Contains the interface to the parser. |

```
$ calc
p=3.14159
r=2.5
p*r*r
=19.6349
```

Figure 1. Sample Dialog using program calc
Note: Ctrl-D is the end-of-file marker for the keyboard

In this discussion, we focus on yacc directives:

- %union
- %token
- %type

We also discuss the semantic stack record *yylval* and its interaction with yacc pseudo variables, $$, $1, $2 … .

File mkcalc

```
rm calclex.cpp
rm calcpar.cpp
rm *.o
rm calc
make -f makecalc
```

Figure 2. File mkcalc

Notes:

1. Files calclex.cpp and calcpar.cpp are created in the process of making executable file calc. We want to remove both files before we create a new version of the calculator.
2. Likewise, we wish to remove all object files and the executable file, calc, before we create a new version.
3. The command make -f makecalc creates a new version of the calculator, calc.

File makecalc

```
#-------------------------------------------------------------------
# File makecalc creates a calculator as described in Chapter 3 of
# Lex & Yacc by Mssers. Levine, Mason, and Brown.
#-------------------------------------------------------------------
# Author:   Thomas R. Turner
# E-Mail:   trturner@uco.edu
# Date:     May, 2020
#-------------------------------------------------------------------
# Copyright May, 2020 by Thomas R. Turner.
# Do not reproduce without permission from Thomas R. Turner.
#-------------------------------------------------------------------
# Object files
#-------------------------------------------------------------------
obj     =       calcpar.o \
                calclex.o \
                calc.o
#-------------------------------------------------------------------
# Bind the subset Pascal Parser and Scanner
#-------------------------------------------------------------------
calc:           ${obj}
                g++ -o calc ${obj} -lm -ll
#-------------------------------------------------------------------
# File calc.cpp contains main
#-------------------------------------------------------------------
calc.o:         calc.cpp calcpar.h
                g++ -c -g calc.cpp
```

Figure 3. File makecalc

```
#-------------------------------------------------------------------
# Create file calclex.cpp from file calclex.l using lex.
# Rename the generate C-file lex.yy.c to calclex.cpp
# in preparation for compiling it with the C++ compiler
#-------------------------------------------------------------------
calclex.cpp:    calclex.l calclex.h
                lex calclex.l
                mv lex.yy.c calclex.cpp
#-------------------------------------------------------------------
# Compile the lexer, calclex.cpp
#-------------------------------------------------------------------
calclex.o:      calclex.cpp calclex.h y.tab.h
                g++ -c -g calclex.cpp
#-------------------------------------------------------------
# Create files y.tab.c and y.tab.h from file calcpar.y
# Rename file y.tab.c to calcpar.cpp
# in preparation for compiling it with the C++ compiler
#-------------------------------------------------------------
calcpar.cpp:    calcpar.y
                yacc -d -v calcpar.y
                mv y.tab.c calcpar.cpp
#-------------------------------------------------------------
# Compile the parser, calcpar.cpp
#-------------------------------------------------------------
calcpar.o:      calcpar.cpp calcpar.h calclex.h y.tab.h
                g++ -c -g calcpar.cpp
#-------------------------------------------------------------
#-------------------------------------------------------------
```

Figure 3. File makecalc (continued)

Notes:

1. Comments begin with # and end with a newline.
2. A \ concatenates the following line with the current line.
3. The symbol **obj** contains a list of the object files created by this makefile.
4. The format of a makefile directive is:

```
soup:   ingredients
\t      instruction 1
\t      instruction 2
…
\t      instruction n
```

   Example:

```
calc.cpp:   calc.cpp calcpar.h
\t          g++ -c -g calc.cpp
```

   The goal is to create relocatable object file calc.o. Files calc.cpp and any include-files referenced is the source calc.cpp are listed as necessary "ingredients."
   The tab character symbolized by \t is given to visualize an essential character that is invisible. Please note that forgetting to put a **tab** character on this line is likely the most common and **frequent error** when constructing makefiles. The command:

```
g++ -c -g calc.cpp
```

creates the relocatable file calc.o.

File calc.cpp

```
//-----------------------------------------------------------------
//File calc.cpp is the main entry for a four-function calculator
//described by Levine, Mason, and Brown in chapter 3 of Lex & Yacc.
//-----------------------------------------------------------------
//Author:   Thomas R. Turner
//E-Mail:   trturner@uco.edu
//Date:     May, 2020
//-----------------------------------------------------------------
//Copyright May, 2020 by Thomas R. Turner
//Do not reproduce without permission from Thomas R. Turner
//-----------------------------------------------------------------
//C++ Standard include files
//-----------------------------------------------------------------
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdio>
#include <string>
using namespace std;
//-----------------------------------------------------------------
//Application include files
//-----------------------------------------------------------------
#include "calclex.h"
#include "calcpar.h"
//-----------------------------------------------------------------
//-----------------------------------------------------------------
void CalcMgr(void){int rc=yyparse();}
//-----------------------------------------------------------------
//Function main processes command line arguments
//-----------------------------------------------------------------
int main()
{   CalcMgr();
    return 0;
}
```

Figure 4. File calc.cpp

Notes:

1. Please observe that function main does little more than invoke the parser, *yyparse*().

File calclex.l

```
%{
//--------------------------------------------------------------------
//File calclex.l contains a scanner specification for a calculator
//defined by Messrs. Levine, Mason, and Brown.
//--------------------------------------------------------------------
//Authors:      J. R. Levine, T. Mason, and D. Brown
//Revised by:   Thomas R. Turner
//E-Mail:       trturner@uco.edu
//Date:         May, 2020
//--------------------------------------------------------------------
//Copyright May, 2020 by Thomas R. Turner.
//Do not reproduce without permission from Thomas R. Turner.
//--------------------------------------------------------------------
//C++ inlcude files
//--------------------------------------------------------------------
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <cstdlib>
using namespace std;
//--------------------------------------------------------------------
//Application include files
//--------------------------------------------------------------------
#include "calclex.h"
#include "y.tab.h"
//--------------------------------------------------------------------
//Functions
//--------------------------------------------------------------------
int Identifier(void);
int Real(void);
//--------------------------------------------------------------------
//Externals
//--------------------------------------------------------------------
extern double vbltable[26];
%}
```

Figure 5. File calclex.l (definition section)

Notes:

1. The definition section is everything before the first %% that you can see on the following page.
2. The most important part of the definition section is the literal block enclosed in **%{ … %}**.
3. The copyright notice is given to prohibit you from using this example if you, later, become an instructor, without obtaining my permission first.
4. Note the inclusion of file y.tab.h. File y.tab.h is created by the parser generator *yacc*. File y.tab.h assigns positive integer codes to all terminal and nonterminal symbols.
5. Functions that are called from the rules section must be declared in the literal block.

6. Any data declared in another compilation unit – another file – that is referenced in this file must be declared in the literal block.

File calclex.l (continued)

```
%%
[ \t];                                /* ignore whitespace*/
[\n\r]                                return CRNL;
[a-z]                                 return Identifier();
[0-9]+                                return Real();
[0-9]+[Ee][+-]?[0-9]+                 return Real();
[0-9]+\.[0-9]+([Ee][+-]?[0-9]+)?      return Real();
"+"                                   return PLUS;
"-"                                   return MINUS;
"*"                                   return MUL;
"/"                                   return DIV;
"("                                   return LPAREN;
")"                                   return RPAREN;
"="                                   return ASSIGN;
```

Figure 5. File calclex.l (rules section) (continued)

Notes:

1. Each rule in the rules section begins with a regular expression and is followed by an action written in C or C++. If the action is a single statement, it must be terminated with a semicolon. Multiple statements must be enclosed in a block.
2. Capitalized names are symbols for integer codes assigned to those symbol in file y.tab.h.

File calclex.l (continued)

```
%%
//------------------------------------------------------------------
//User subroutines
//------------------------------------------------------------------
//------------------------------------------------------------------
//Function Identifier stores the index of the single letter identifier
//in yacc semantic structure yylval.vblno.
//------------------------------------------------------------------
int Identifier(void)
{    yylval.vblno=yytext[0]-'a';
     return NAME;
}
//------------------------------------------------------------------
//Function Real converts the numeric string to a double and stores it
//in yacc semantic structure yylval.dval
//------------------------------------------------------------------
int Real(void)
{    yylval.dval = atof(yytext);
     return NUMBER;
}
```

Figure 5. File calclex.l (user subroutine section) (continued)

Notes:

1. Member *vblno* (variable number) in the structure *yylval* is assigned an integer value, $0 \leq i \leq 25$. The %union directive in file calcpar.y defines all members of structure *yylval*.
2. Member *dval* in the structure *yylval* is assigned the floating-point equivalent of the string recognized by the scanner (*yylex***()**).
3. Together *vblno* and *dval* represent a primitive symbol table consisting of floating-point variables having names 'a', 'b', … 'z';

File calclex.h

```
#ifndef calclex_h
#define calclex_h 1
//--------------------------------------------------------------------
// File calclex.h defines class Lexer.
//--------------------------------------------------------------------
// Author: Thomas R. Turner
// E-Mail: trturner.uco.edu
// Date:   May, 2020
//--------------------------------------------------------------------
// Copyright May, 2020 by Thomas R. Turner
// Do not reproduce without permission from Thomas R. Turner.
//--------------------------------------------------------------------
//--------------------------------------------------------------------
// Standard C and C++ include files
//--------------------------------------------------------------------
#include <cstdio>
#include <fstream>
#include <iostream>
//--------------------------------------------------------------------
//Namespaces
//--------------------------------------------------------------------
using namespace std;
//--------------------------------------------------------------------
//Function: yylex
//Function yylex is the calcner.  Function yylex returns an integer
//token code as defined above or 0 if end-of-file has been
//reached.
//--------------------------------------------------------------------
#ifdef __cplusplus
extern "C"
#endif
int yylex (void);
#endif
```

Figure 6. File calclex.h

Notes:

1. Function *yylex* is generated by the linux utility lex. Since *yylex* is a C-function rather than a C++ function its name is not mangled and must be so designated when compiled by the C++ compiler.

File calcpar.y

```
%{
//-------------------------------------------------------------------
//File calc.y contains a calculator grammar
//defined by Messrs. Levine, Mason, and Brown.
//-------------------------------------------------------------------
//Authors:      Levine, Mason, and Brown
//Revised by:   Thomas R. Turner
//E-Mail:       trturner@uco.edu
//Date:         May, 2020
//-------------------------------------------------------------------
//Copyright May, 2020 by Thomas R. Turner.
//Do not reproduce without permission from Thomas R. Turner.
//-------------------------------------------------------------------
//C++ inlcude files
//-------------------------------------------------------------------
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
using namespace std;
//-------------------------------------------------------------------
//-------------------------------------------------------------------
//Application include files
//-------------------------------------------------------------------
#include "calcpar.h"
//-------------------------------------------------------------------
//Functions
//-------------------------------------------------------------------
void yyerror(const char* m);
//-------------------------------------------------------------------
//Externals
//-------------------------------------------------------------------
//-------------------------------------------------------------------
//File Global Variables
//-------------------------------------------------------------------
double vbltable[26];
//-------------------------------------------------------------------
%}
```

Figure 7. File calcpar.y (literal section)

Notes:

1. File calcpar.y contains the specification for the parser formally called a syntax analyzer. By translating this file using the *yacc* tool an LL parser is created. The principal advantage of the *yacc* tool is that the grammar is specified in the ordinary way with few syntactic differences to accommodate computer character sets.
2. Like file calclex.l, functions called in the rules section and variables referenced in the rules section must be declared in the literal block. Note that array *vbltable* (variable

table) is defined in this compilation unit meaning that storage is allocated for *vbltable* in this compilation unit.

File calcpar.y (continued)

```
%union {
    double dval;
    int vblno;
}
%token              CRNL
%token              ASSIGN
%token              PLUS
%token              MINUS
%token              MUL
%token              DIV
%token              LPAREN
%token              RPAREN
%token    <vblno>   NAME
%token    <dval>    NUMBER
%type     <dval>    expression
%type     <dval>    primary
%type     <dval>    factor
```

File calcpar.y (definition section continued)

Notes:

1. The **%union** directive defines members used to perform semantic computations while the program is being parsed. It is this feature that gives rise to the phrase "*syntax-directed translation*".
2. We define two members *dval* and *vblno*.
   2.1. Symbol *dval* defines the semantic type associated with expressions. Expressions are limited to those values that can be represented by a floating-point value, i.e. numeric values can be expressed as floating-point values.
   2.2. Symbol *vblno* defines the semantic type associated with NAMEs. NAMEs are converted to an index into the primitive symbol table. Identifier *vblno* is the integer index into the symbol table, *vbltable*.
3. The directive **%token** defines terminal symbols. In effect, identifiers *CRNL*, *ASSIGN*, …, and *NUMBER* are all terminal symbols.
4. We have two versions of the directive **%token**. The first version is type less and the second version assigns a type to the terminal symbol.
   4.1. Terminal symbol *NAME* is given an associated semantic value, <vblno>, as an index into the symbol table, *vbltable*.
   4.2. Terminal symbol *NUMBER* is given an associated semantic value, <*dval*>, as a numeric value containing the value to the string representation of the *NUMBER*.
5. The directive **%type** associates a type with a nonterminal symbol. In the case of the directive

   **%type** **<***dval***>** *expression*

   the nonterminal *expression* is given a floating-point value (**double**) associated with it. The symbol **<***dval***>** and its prior declaration in the %union directive define **<***dval***>** as a synonym for C++ type **double**.

File calcpar.y (continued)

```
%%
statement_list:
   statement CRNL
   {
   }
statement_list:
   statement_list statement CRNL
   {
   }
statement:
   NAME ASSIGN expression
   {
      vbltable[$1]=$3;
   }
statement:
   expression
   {
      cout << "=" << ($1) << endl;
   }
expression:
   expression PLUS factor
   {
      $$ = $1 + $3;
   }
expression:
   expression MINUS factor
   {
      $$ = $1 - $3;
   }
expression:
   factor
   {
      $$ = $1;
   }
```

File calcpar.y (rules section)

Notes:

1. The format of the rules section is:

   *rule action*

   where rule is a grammar rule formatted as

   leftside: rightside

   For example, the rule *expression* → *expression* **+** *factor* is expressed

   expression: expression PLUS factor

   Nonterminal symbols are coded in all lowercase letters by convention. Terminal symbols are capitalized by convention. The scanner establishes the relation between the terminal symbol PLUS and the character '+'.

   An action is simply a block of C++ statements enclosed in curly braces.

2. In the rule *statement*: *NAME ASSIGN expression*, $$ refers to *statement*, $1 refers to *NAME*, $2 refers to *ASSIGN*, and $3 refers to *expression*.

   The action *vbltable*[**$1]=$3;** means that *NAME*, or **$1**, is used as an index into the array vbltable. We know, from our previous discussion, that *NAME* has been associated with the type *vblno* and, further, <*vblno*> is a synonym for the type integer. That means that **$1** has type integer also.

   What integer value does **$1** have? Recall that *NAME*s are identifier that are limited to single lowercase letters. The scanner mapped the single letters to the integers 0 – 25 in the statement *yylval*.*vblno*=*yytext*[**0]-'a'.** The pseudo variable **$1** has an integer value ranging from 0 to 25.

   Every time variable *a* appears in calculator expressions it is translated to *vbltable*[**0**]. Likewise, for remaining single letter variables *b*, *c*, …, *z*, they are translated to *vbltable*[**1**], *vbltable*[**2**], …, *vbltable*[**25**] respectively.

   In a similar way pseudo variable **$3** is associated with *expression*, it being the third grammar symbol in the rule. Grammar symbol expression has been given type <*dval*>, a synonym for type **double**. We know that array *vbltable* is compatible with type double from its declaration, **double** *vbltable*[**26];.**

   The action, *vbltable*[**$1]=$3;**, stores the value of the expression in the storage location allocated for the variable.
3. By default, if no action is specified, the action, **$$=$1**, is executed.
4. Whenever the parser reduces a rule, it executes user C or C++ code associated with the rule, known as the rule's *action*. The action appears in braces after the end of the rule, before the semicolon or vertical bar. The action code can refer to the values of the right-hand side symbols as **$1**, **$2**, …, and can set the value of the left-hand side by setting **$$**.

   In the rules defining expression, the $-variables act like structures. For example, in the rule:

   *expression*: *expression* PLUS *expression* {$$ = $1 + $3;}

   what is really happening is:

   *expression*: *expression* PLUS *expression* {$$.dval = $1.dval + $3.dval;}

File calcpar.y (continued)

```
factor:
   factor MUL primary
   {
      $$ = $1 * $3;
   }
factor:
   factor DIV primary
   {
      if ($3 == 0.0)
        yyerror("divide by zero");
      else
        $$ = $1 / $3;
   }
factor:
   primary
   {
      $$ = $1;
   }
primary:
   MINUS primary
   {
      $$ = -$2;
   }
primary:
   LPAREN expression RPAREN
   {
      $$ = $2;
   }
primary:
   NUMBER
   {
      $$ = $1;
   }
primary:
   NAME
   {
      $$ = vbltable[$1];
   }
```

File calcpar.y (rules section continued)

File calcpar.y (continued)

```
%%
//-------------------------------------------------------------------
//User function section
//-------------------------------------------------------------------
void yyerror(const char* m)
{     cout << endl;
      cout << m;
      cout << endl;
}
//-------------------------------------------------------------------
//-------------------------------------------------------------------
```

File calcpar.y (user subroutine section)

Notes:

1. Since function *yyerror* is not supplied, the programmer must create a function *yyerror*

File calcpar.h

```
#ifndef calcpar_h
#define calcpar_h 1
//-----------------------------------------------------------------
// File calcpar.h defines the interface to the parser generated by
// yacc.
//-----------------------------------------------------------------
// Author: Thomas R. Turner
// E-Mail: trturner.uco.edu
// Date:   May, 2020
//-----------------------------------------------------------------
// Copyright May, 2020 by Thomas R. Turner
// Do not reproduce without permission from Thomas R. Turner.
//-----------------------------------------------------------------
//Application include files
//-----------------------------------------------------------------
#include "calclex.h"
//-----------------------------------------------------------------
//-----------------------------------------------------------------
//Function yyparse is the parser generated by yacc.
//-----------------------------------------------------------------
#ifdef __cplusplus
extern "C"
#endif
int yyparse (void);
#endif
```

Figure 8. File calcpar.h

Notes:

1. Function *yyparse* is generated by the linux utility yacc. Since *yyparse* is a C-function rather than a C++ function its name is not mangled and must be so designated when compiled by the C++ compiler.