CSCI 447 – Spring 2002
Yacc Assignment

Professor: Muhammed Mudawwar
Due Date: Monday, May 20, 2002

Objectives

1- To develop a Yacc specification for a given grammar.
2- To translate into 3-address code.

The M Language

The language that will be used is the same M language that was used in the previous assignment. Due to time constraint, record types are eliminated from the language and grammar rules and only one-dimensional arrays are supported.

Grammar Rules

The following is the grammar of the M language:

```
Program    → { ConstDecl | VarDecl | FuncDef }
ConstDecl  → const Decl ':=' Expr ';'
VarDecl    → var Decl [':=' Expr] ';'
Decl       → idList ':' TypeExpr
idList     → id {',' id}
TypeExpr   → int [Dim] | real [Dim] | char [Dim] | bool [Dim]
Dim        → '[' intconst ']'
FuncDef    → function id '(' [FormalList] ')' [':' TypeExpr] Block
FormalList → Formal {',' Formal}
Formal     → [const] Decl | var Decl
Block      → '{' {stmt} '}'
Stmt       → ConstDecl
           | VarDecl
           | Object ':=' Expr ';'
           | if Expr Block {else if Expr Block} [else Block]
           | while Expr Block
           | return Expr ';'
           | FuncCall ';'
FuncCall   → id '(' [ExprList] ')'
ExprList   → Expr {',' Expr}
Object     → id [Suffix]
Suffix     → '[' Expr ']'

Expr        →  Expr or Expr
Expr        |  Expr and Expr
Expr        |  Expr relop Expr
Expr        |  Expr addop Expr
Expr        |  Expr mulop Expr
Expr        |  UnaryOp Expr
Expr        |  Object
```

```
              |   intconst
              |   realconst
              |   charconst
              |   boolconst
              |   strconst
              |   FuncCall
              |   '(' Expr ')'
UnaryOp    → addop | not
```

Yacc Specification and Operator Precedence

Convert the above grammar into a conflict-free Yacc specification. Notice that the grammar rules for *Expr*ession are ambiguous. Therefore, you should remove this ambiguity by defining the precedence and associativity of operators. The **or** operator has the least precedence, then the **and** operator, then the relational operators **relop** (<, <=, >, >=, =, <>), then the **addop** operators (+, –), then the **mulop** operators (*****, /, **mod**), then the *unary op*erators (+, –, **not**). Most operators are left associative. However, the unary operators are right associative and the relational operators are non-associative.

Syntax Error Handling

If a syntax error occurs while parsing, a meaningful error message should be reported specifying the line number and character position. Compilation can terminate at the first syntax error. Error recovery is not required. However, you are invited to recover from syntax errors if you have time.

Symbol and Literal Tables

Use the same symbol and literal tables that you developed for the previous assignment. Make sure to use appropriate actions to enter identifiers into the identifier table. Due to time constraints, do not use the type table and do not enter types. Furthermore, you will not need the field table because records and fields are dropped from the grammar.

Static Semantics Checking:

Due to time constraints, only the following static semantic rules should be checked. Error messages should be reported when these rules are violated, but parsing and translation should continue.

1 All variables must be declared before they can be used. Functions should be defined before they can be called. The *input* and *output* function symbols should be initially inserted into the identifier table to locate them when they are called.

2 Identifiers cannot be declared more than once within the same scope. However, the same identifier name can appear in different scopes. Function names cannot be overloaded (with the exception of the predefined *input* and *output* functions, which can accept an arbitrary number and types of parameters).

3 Formal **const** parameters can be read only inside a function. They cannot appear on the left hand side of an assignment statement. Furthermore, **const** parameters, **const** identifiers, and literals cannot be passed to **var** parameters. Formal **var** parameters must receive **var** identifiers as their actual parameters. **Var** parameters can be read and written and can be passed as actuals to formal **const** parameters.

Translation into Three-Address Code

Here are the three-address instructions needed to translate programs written in the M-language:

1 Assignment instructions of the form: **`x := y op z`**, where *op* is a binary arithmetic or a logical operation.

2 Assignment instructions of the form: **`x := op y`**, where *op* is a unary operation

3 Copy statements of the form: **`x := y`**

4 Variables and literal constants in three-address instructions are entries in symbol and literal tables.

For example, the assignment statement:

`a := not (b > 'A' and c < 3)` is translated into:

```
(TEMP 1) := (VAR b LOCAL) > (LIT 'A')
(TEMP 2) := (CONST c PARAM) < (LIT 3)
(TEMP 3) := (TEMP 1) AND (TEMP 2)
(TEMP 4) := NOT (TEMP 3)
(VAR a GLOBAL) := (TEMP 4)
```

where *a* is global variable, *b* is a local variable, and *c* is a constant parameter. (*TEMP i*) is a compiler-generated temporary. Temporaries are numbered starting at 1.

The following is a list of operators that can appear in a source file:

```
+     -     *     /     mod   or    and   not
<     <=    >     >=    =     <>
```

5 Unconditional jumps are of the form: **`GOTO (LABEL L)`**. A label **`(LABEL L)`** can precede instructions.

6 Conditional jumps are of the form:

`IF x GOTO (LABEL L)` or
`IFNOT x GOTO (LABEL L)`.

The **if** and **while** statements are translated using the conditional and unconditional jumps. For example, the following **if** statement:

```
if x >= y { max := x; } else { max := y; }
```

is translated into:

```
(TEMP 1) := (CONST x PARAM) >= (CONST y PARAM)
IFNOT (TEMP 1) GOTO (LABEL 1)
(VAR max PARAM) := (CONST x PARAM)
GOTO (LABEL 2)
(LABEL 1)
(VAR max PARAM) := (CONST x PARAM)
(LABEL 2)
```

The following **while** statement:

```
i := 1;
while i <= n do { sum := sum+i; i:= i+1; }
```

is translated into:

```
(VAR i LOCAL) := (LIT 1)
GOTO (LABEL 4)
(LABEL 3)
(VAR sum LOCAL) := (VAR sum LOCAL) + (VAR i LOCAL)
(VAR i LOCAL) := (VAR i LOCAL) + (LIT 1)
(LABEL 4)
(TEMP 1) := (VAR i LOCAL) <= (CONST n PARAM)
IF (TEMP 1) GOTO (LABEL 3)
```

7 For array indexing, we will need the following 3-address instructions:

```
x := y [] i
x [] i := y
```

The *opcode* of the first instruction is **`:=[]`**, while the *opcode* of second is **`[]:=`**.

For example, **`b[2*i] := a[j+3] * c`** is translated into:

```
(TEMP 1) := (VAR j LOCAL) + (LIT 3)
(TEMP 2) := (CONST a PARAM) [] (TEMP 1)
(TEMP 3) := (TEMP 2) * (VAR c LOCAL)
(TEMP 4) := (LIT 2) * (VAR i LOCAL)
(VAR b PARAM) [] (TEMP 4) := (TEMP 3)
```

8 To translate functions and calls, we need a **FUNCTION** entry label, an **ARG** instruction for passing actual arguments to a function, a **CALL** instruction for making function calls, and a **RETURN** instruction to return from a function call.

For example, the following function:

```
function f(m,n:int):int {
  if n <= 0 { return 1; }
  else {
    var k:int := m/2;
    k := f(n,k);
    return k;
  }
}
```

is translated into:

```
(FUNCTION f)
(TEMP 1) := (CONST n PARAM) <= (LIT 0)
IFNOT (TEMP 1) GOTO (LABEL 1)
RETURN (LIT 1)
(LABEL 1)
(TEMP 2) := (CONST m PARAM) / (LIT 2)
(VAR k LOCAL+1) := (TEMP 2)
ARG (CONST n PARAM)
ARG (VAR k LOCAL+1)
(VAR k LOCAL+1) := CALL (FUNCTION f)
RETURN (VAR k LOCAL+1)
```

Report:

A detailed report should be written explaining your design and implementation. Specifically, you need to discuss the Yacc specification, grammar attributes, semantic actions, and the data structures used for the implementation of the 3-address code.

To Submit:

1 The report document.

2 A floppy disk containing all your files (lex files, C/C++ files, executable program, test inputs and outputs).

Grading

Your grade will be divided into the following components:

1 Correctness and output
2 Yacc specification, semantic attributes, and semantic actions
3 Intermediate code, implementation details, and documentation
4 Report Document