Compilers and Interpreters
Tutorial 2

Sorin Manolache, sorma@ida.liu.se

- online version at http://www.ida.liu.se/~sorma/teaching/compinterp/tutorial2.pdf

Overview

- lab 3 – to generate a parser for a Pascal-like language using the *Bison* parser generator
- a forest of *abstract syntax trees* (AST) is constructed as result of the parsing (one tree per function/main program)
- lab4 – to generate intermediate code for the ASTs

Bison

- a parser generator
- input: a file (specification file) containing mainly the grammar definition
- output: a C source file containing the parser
- the entry point is the function

```
int yyparse();
```

- `yyparse` reads tokens by calling `yylex` and parses until
  - end of file to be parsed, or
  - unrecoverable syntax error occurs
- returns 0 for success and 1 for failure

Bison – Input File

- 3 parts separated by %% on a new line

```
optional definitions
%%
optional rules
%%
optional additional C code
```

- rules section
  - format

```
<nonterm>:<comp1> {<opt ac>} <comp2> {opt ac>} ...
    ;
```

  - `comp` denotes the terminals or non-terminals that compose the `nonterm` on the left

Bison – Semantic Attributes

- both terminals and non-terminals may have one semantic attribute attached to it
- if there are semantic attributes of different types, the possible types have to be enumerated (YYSTYPE) and the mapping specified between terminals and non-terminals on one part and the semantic attribute types on the other part

Example

```
%union {
   int i_val;
   double r_val;
}
%token <i_val> I_NUM
%token <r_val> R_NUM
%type <r_val> expression term factor
%%
expression : expression ‘+’ term
           {
              $$ = $1 + $3;
           }
           ;
factor : I_NUM
           {
              $$ = (double)($1);
           }
```

Mid-Rule Actions

```
stmt: LET '(' var ')'
        { $<context>$ = push_context ();
          declare_variable ($3); }
      stmt { $$ = $6;
             pop_context ($<context>5); }
      ;
```

- a mid-rule action is a *anonymous* component
- $$ in a mid-rule denotes the semantic attribute of itself
- being anonymous, one cannot specify in the declaration section the mapping between it and its type
- therefore, it has to be done *in situ*:

```
$<type>$ =
= $<type>n
```

Syntax Errors

- the specification can be stuffed with error productions
- they help the compiler to recover from syntax errors and to continue to parse
- example:

```
functionCall : ID '(' paramList ')'
             |
               ID '(' error ')'
             ;
```

Synthesized and Inherited Attributes

```
expr : term /* send $1 as an arg to e_prime */ e_prime
            { $$ = $2; }
            ;
e_prime : ‘+’ /* send the arg as an arg to e_prime */
              e_prime
            { $$ = arg + $2; }
          |
            { $$ = arg; }
          ;
```