CS24210 Syntax Analysis and Topics in Programming Languages

Edel Sherratt

Revision

- Perl - Egbert
- lex and yacc - a little language

Perl - Egbert

```
sub shoppingList {

  local ($order, $shopping_list) = @_;
    # $shopping_list is a reference to %shopping_list

  local %price = (brassica=>30, book=>100, fruit=>20,
    hot_drink=>50, newspaper=>45);

  $order =~ tr/A-Z/a-z/;

  local $category;

  if ($order =~ /cabbage|kale|green|broccoli/) {
    $category = 'brassica';
  } elsif ($order =~ /book/) {
    $category = 'book';
  } elsif ($order =~ /newspaper/) {
    $category = 'newspaper'
  } elsif ($order =~ /tea|coffee/) {
    $category = 'hot_drink';
  } elsif ($order =~ /apple|banana|pear/) {
    $category = 'fruit';
  }
```

```
  if ($category) {
    $shopping_list{$&} += $price{$category};
    shoppingList  ($‘.$’,\%shopping_list);
    #notice the reference to %shopping_list
  } else {
    local $total, $return_string;
    foreach(keys(%$shopping_list)) {
      $return_string = $return_string.
        p("Your $_ will cost ".$shopping_list{$_}.
         " pence");
      $total += $shopping_list{$_};
    }
    $return_string =
      $return_string.p("Total bill: $total pence");
    return $return_string;
  }

}
```

Perl – Egbert

```
sub addItem {

  local ($before, $item, $category, $shopping_list)
    = @_;
  $shopping_list{$item} += $price{$category}
    unless
    ($before =~ /.*(no|don\'t\s*want\s*(a|any))\s*$/);

}

sub shoppingList {

... as before

if ($category) {
    #$shopping_list{$&} += $price{$category};
    addItem ($`,$&,$category,\%shopping_list);

... as before
}
```

Grammar for a little language

```
program -> sequ
sequ    -> command | command ; sequ
command -> ID := expr | ID += expr
| ID -= expr
| if expr then sequ else sequ endif
| while expr do sequ endwhile
expr    -> NUMBER | ID | expr < expr
```

yacc specification

```
... declarations of tokens, constants, etc.
%%
program : sequ END   { preorder($$.treeptr,visitor); }
        ;
sequ    : command
        | command SEM sequ  { $$.treeptr = mknode(
                              makeLabel(seq_label,0),
                              $1.treeptr,$3.treeptr); }
        ;
```

```
command : ID ASSIGN expr    { $$.treeptr = mknode(
                        makeLabel(assign_label,$1.value),
                        mkleaf(
                         makeLabel(id_label,$1.value)),
                                $3.treeptr); }
        | ID ADDTO expr     { ... like assign
        | ID SUBFROM expr   { ... like assign
        | IF expr THEN sequ ELSE sequ ENDIF
                  { $$.treeptr = mknode(
                      makeLabel(if_label,0),
                        $2.treeptr,
                        mknode(
                          makeLabel(branch_label,0),
                            $4.treeptr,$6.treeptr)); }
        | WHILE expr DO sequ ENDWHILE
                        { $$.treeptr = mknode(
                           makeLabel(while_label,0),
                           $2.treeptr,$4.treeptr); }
```

```
expr    : NUMBER  { $$.treeptr = mkleaf(
                    makeLabel(number_label,$1.value)); }
        | ID      { $$.treeptr = mkleaf(
                      makeLabel(id_label,$1.value)); }
        | expr LT expr  { $$.treeptr = mknode(
                      makeLabel(lt_label,0),
                      $1.treeptr, $3.treeptr); }
        ;
%%
... include lex.yy.c ... declare useful functions
```

lex specification

```
letter  [A-Za-z]
digit   [0-9]

%%

":="            {return ASSIGN;}
"+="            {return ADDTO;}
"<"             {return LT;}
[;]             {return SEM;}

"if"            {return IF;}
"then"          {return THEN;}
"else"          {return ELSE;}
"endif"         {return ENDIF;}

"while"         {return WHILE;}
"do"            {return DO;}
"endwhile"      {return ENDWHILE;}

[$]             {return END;}

[+-]?{digit}+   {sscanf(yytext,"%d",&(yylval.value));
                return NUMBER;}
{letter}({letter}|{digit})* {yylval.value = installId();
                return ID;}
%%
... definition of installID
```