CS 2412 Data Structures

Chapter 1

Case Study: The Polish Notation

The problem:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

C formula: `(-b + (b**2-4.0*a*c)**(0.5))/2.0*a`

How the complier handle the formula?

Operators and priorities

operators	priority
`** ! f`, unary operators	6
* / %	5
`+ -`(binary)	4
`== !=`	3
`< <= > >=`	2
`&& ||`	1
`=`	0

Main idea

- Expression trees
- How to evaluation of an expression tree (p.537 Fig 12.1)
- postorder traversal
- Polish notation: postfix form

Note: C compiler does not use this method.

Evaluation of postfix expressions

Define

- `token` to be a single operator or operand
- `void GetToken(Token token, Expression expr)`
- function `Kind(  )` with a return enumerated type `OPERAND, UNARYOP, BINARYOP, ENDEXPR`
- `Value DoUnary(Token token, Value x); Value DoBinary(Token token, Value x, Value y); Value GetValue(Token token);`

```
Value EvaluatePostfix(Expression expr)
{
 KindType type;
 Token token;
 Value x,y;
 Stack stack;
 CreateStack(&stack);
 do {
   GetToken(token, expr);
   switch (type = Kind(token)) {
   case OPERAND:
     Push(GetValue(token),&stack);
     break;
   case UNARYOP:
     Pop(&x, &stack);
     Push(DoUnary(token, x), &stack);
```

```
      break;
    case BINARYOP:
      Pop(&y, &stack);
      Pop(&x, &stack);
      Push(DoBinary(token, x, y),&stack);
      break;
    case ENDEXPR:
      Pop(&x, &stack);
      if(!StackEmpty(&stack))
       error("Incorrect expression");
      break;
    }
  }while (type != ENDEXPR);
  return x;
}
```

Correctness of the algorithm

For a sequence E of operands, unary operators, and binary operators, form a running sum by starting at the left end of E and counting $+1$ for each operand, 0 for each unary operator, and -1 for each binary operator. E satisfies the **running-sum condition** provided that this running sum never falls below 1, and is exactly 1 at the right-hand end of E.

(p. 544 Fig 12.2)

Theorem If E is a properly formed expression in postfix form, then E must satisfy the running sum condition.

Theorem A properly formed expression in postfix form will be correctly evaluated by `EvaluatePostfix`.

Theorem If E is any sequence of operands and operators that satisfies the running-sum condition, then E is a properly formed expression in postfix form.

Translation from infix form to postfix form:

First we assume that no unary operators that are placed to the right of their operand. The the idea is delaying each operator until its right-hand operand has been translated.

Example:

Infix form: `x+y x+y*z`

Postfix form: `xy+ xyz*+`

From the second expression, we see that we must take both parentheses and priorities of operators into account.

If *op* is an operator in an infix expression, then its right-hand operand contains all token on its right until one of the following is encountered:

1. the end of the expression;
2. an unmatched right parenthesis ‘)’;
3. an operator of priority less than or equal to that of *op*, and not within a bracketed sub-expression, if *op* has priority less than 6; or
4. an operator of priority strictly less than that of *op*, and not within a bracketed subexpression, if *op* has priority 6.

Use a stack to implement:

1. At the end of the expression, all operators are output
2. A right parenthesis causes all operators found since the corresponding left parenthesis to be output
3. an operator of priority not 6 causes all other operators of greater or equal priority to be output
4. An operator of priority 6 causes all other operators of strictly greater priority to be output, if such operators exist.