ECE 2574: Data Structures and Algorithms - Applications of Recursion I

C. L. Wyatt

Today we will look a common task that is easily solved using a recursive solution, parsing algebraic expressions.

- Warmup
- Review of algebraic expressions
- converting between prefix and postfix expressions
- an implementation in C++

Why prefix or postfix notation?

Many of you asked, in some form or another, why prefix or positfix notation?

- unambiguous, there are no operator precedence rules
- easy to parse (translate into a tree form) for evaluation
- supports operators with n-ary arguments with no additional syntax

For this reason, prefix and postfix notation is used in many programming and data description languages

- languages in the Lisp family use prefix notation
- stack-based languages generally use postfix notation

Algebraic Expressions

Lets say we are going to write a program to act as a calculator. For example:

```
(a + b)*c
(a/b)*c
(a-b-c-d)/e
```

How does the calculator decide if the expression is valid ?

Lets start with a less complex Algebraic grammar

Prefix expressions. In prefix notation the operation is written first, followed by the two operands.
Examples:

- `* + a b c` in infix notation is `(a + b)*c`
- `+ / a b - c d` in infix notation is `(a / b) + (c - d)`

The grammar looks like:

```
<prefix> = <operand> | <operator> <prefix> <prefix>
<operator> = + | - | * | /
<operand> = a | b | c | .... | z
```

Validation of a prefix expression using recursion

```
<prefix> = <operand> | <operator> <prefix> <prefix>
<operator> = + | - | * | /
<operand> = a | b | c | .... | z
```

Base step is simple: check for operator at string beginning.
The recursive step is a little more complicated. The key is that if `<prefix>` is a valid prefix `<prefix><ch>`, where `<ch>` is any non-blank character, is not.

Validation of a prefix expression using recursion

```
function endPre(in s:string, in first:int): int

last = s.length() - 1
if( first < 0 or first > last )
    return -1

ch = first char of s
if(ch is an operand)
    return first
else if(ch is an operator)
    firstEnd = endPre(s, first+1);
    if(firstEnd > -1)
        return endPre(s, firstEnd +1)
    else
        return -1
else
    return -1
endfunction
```

Using the endPre function to validate the grammar

Call endPre at first character in the string
if the last character returned is not the last one it is not a valid prefix expression.

```
function isPre(in s:string): bool

lastChar = endPre(s, 0)

return  lastChar >= 0 AND
        lastChar == s.length()-1

endfunction
```

Warmup #1

Is the following string a valid prefix expression?

`/ + a c d - e g`

False (53% correct)

Similar is the postfix notation

```
<postfix> = <operand> | <postfix> <postfix>
<operator> <operator> = + | - | * | /
<operand> = a | b | c | .... | z
```

Suppose we wanted to convert the prefix expression to a postfix expression.

```
<postfix> = <operand> | <postfix> <postfix> <operator>
<prefix> = <operand> | <operator> <prefix> <prefix>
```

A recursive solution to conversion

```
function convert(in pre:string,
                 out post:string)
ch = first character of pre
delete first character of pre

if ch is an operand
    post = post + ch //concatenate
else
    // recursion to convert 1st
    convert(pre, post)
    // recursion to convert 2nd
    convert(pre, post)
    // concatenate the operator
    post = post + ch
endif

endfunction
```

Validating a postfix expression

```
function endPost(in s:string, in last:int): int

first = 0
if( first > last )
    return -1

ch = last char of s
if(ch is an operand)
    return last
else if(ch is an operator)
    lastEnd = endPost(s, last-1);
    if(lastEnd > -1)
        return endPost(s, lastEnd-1)
    else
        return -1
else
    return -1
endfunction
```

Validating a postfix expression: isPost

Call endPost at last character in the string
if the last position returned is not zero it is not a valid postfix expression.

```
function isPost(in s:string): bool

firstChar = endPost(s, s.length()-1)

return  firstChar == 0

endfunction
```

Warmup #2

Is the following string a valid postfix expression?

```
h r * R f - + t g - e f / *
```

False (49% Correct)

A recursive solution to conversion the other way

```
function convert(in post:string,
                 out pre:string)
ch = last character of post
delete last character of post

if ch is an operand
    pre = pre + ch //concatenate
else
    // concatenate the operator
    pre = pre + ch
    // recursion to convert 1st to temp
    convert(post, temp)
    // recursion to convert 2nd
    convert(post, pre)
    pre = pre + temp // append temp
endif
endfunction
```

Warmup #3

Convert the following prefix expression to a postfix expression.

`+ * A B / C D`

`A B * C D / +` (80 Correct)

Exercise: Implementing and testing in C++

See website.

Next Actions and Reminders

- Read CH pp. 172-186.
- There is no warmup for Monday.
- Program 1 is due Wed at 11:55pm **via Canvas**.
- If you have used all your late days, you **must** turn it in on time.