CSCI 204: Data Structures & Algorithms

Stack Applications

Stack Applications

- Many applications encountered in computer science requires the use of a stack.
  - Balanced delimiters
  - Postfix expressions

Balanced Delimiters

- Many applications use delimiters to group strings of text or simple data into subparts.
  - mathematical expressions
  - programming languages
  - HTML markup

Source Code Example

- Consider the following C source code:

```
int sum_list( int the_list[], int size )
{
  int sum = 0;
  int i = 0;
  while( i < size ) {
    sum += the_list[ i ];
    i += 1;
  }
  return sum;
}
```


Source Code Example

- The delimiters must be paired and balanced.
- We can design and implement an algorithm to:
  - read a C source file, and
  - determine if the delimiters are properly paired.

Valid C Source?

```
from list_stack import Stack

def is_validSource( srcfile ):
  s = Stack()
  for line in srcfile :
    for token in line :
      if token in "{[(" :
        s.push( token )
      elif token in "})]" :
        if s.is_empty() :
          return False
        else :
          left = s.pop()
          if (token == ")" and left != "(") or \
             (token == "]" and left != "[") or \
             (token == ")" and left != "(") :
            return False

  return s.is_empty()
```

stack_apps.py

Mathematical Expressions

- We work with mathematical expressions on a regular basis.
  - Easy to determine the order of evaluation.
  - Easy to calculate.
- But the task is more difficult in computer programs.
  - A program cannot visualize the expression to determine the order of evaluation.
  - Must examine one token at a time.

Types of Expressions

- Three different notations can be used:
  - infix: `A + B * C`
    - Easy for humans, but challenge for program, should we evaluate `A+B` first or `B*C` first?
  - prefix: `+ A * B C`
  - postfix: `A B C * +`
    - Very natural for program to handle

Infix to Postfix

- Infix expressions can be easily converted by hand to postfix notation.

```
A * B + C / D
```

1. Fully parenthesize the expression.

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

2. For each set of (), move operator to the end of the closing parenthesis.

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

Infix to Postfix (cont)

- The expression at the end of step 2:

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

3. Remove all of the parentheses.

```
A B * C D / +
```

- Which results in the postfix version.

The implementation of infix2postfix.py is left as a part of the lab exercise.

Evaluating Postfix Expressions

- We can evaluate a valid postfix expression using a stack structure.
- For each token:
  1. If the current token is an operand, push its value onto the stack.
  2. If the current token is an operator:
     1. pop the top two operands off the stack.
     2. perform the operation (top value is RHS operand).
     3. push the result of the operation back on the stack.
- The final result will be the last value on the stack.

Postfix Evaluation Examples

- To illustrate the use of the algorithm, assume
  - the existence of an empty stack, and
  - the following variable assignments

```
A = 8        C = 3
B = 2        D = 4
```

- Evaluate the valid expression:

```
A B C + * D /
```

Postfix Example #1

Token	Alg Step	Stack	Description
ABC+*D/	1	8	push value of A
A**B**C+*D/	1	8 2	push value of B
AB**C**+*D/	1	8 2 3	push value of C
ABC**+***D/	2(a)	8	pop top two values: y = 3, x = 2
	2(b)	8	compute z = x + y or z = 2 + 3
	2(c)	8 5	push result (5) of the addition
ABC+***D**/	2(a)		pop top two values: y = 5, x = 8
	2(b)		compute z = x * y or z = 8 * 5
	2(c)	40	push result (40) of the multiplication
ABC+***D**/	1	40 4	push value of D
ABC+***D**/	2(a)		pop top two values: y = 4, x = 40
	2(b)		compute z = x / y or z = 40 / 4
	2(c)	10	push result (10) of the division

Postfix Example #2

- What happens if the expression is invalid? A B * C D +

Token	Alg Step	Stack	Description
AB*CD+	1	8	push value of A
A**B***CD+	1	8 2	push value of B
AB***CD+	2(a)		pop top two values: y = 2, x = 8
	2(b)		compute z = x * y or z = 8 * 2
	2(c)	16	push result (16) of the multiplication
AB***C**D+	1	16 3	push value of C
AB***CD+**	1	16 3 4	push value of D
AB*CD**+**	2(a)	16	pop top two values: y = 4, x = 3
	2(b)	16	compute z = x + y or z = 3 + 4
	2(c)	16 7	push result (7) of the addition
Error	***xxxxxx***	***xxxxxx***	***Too many values left on the stack.***