Algebraic expressions

CS110C

Max Luttrell, CCSF

algebraic expressions

- A C++ compiler needs to evaluate algebraic expressions
  - a + b * c
  - a + (b * c)
- We will consider algebraic expressions which use the following four binary operators: +, -, *, /, and parenthesis

algebraic expressions

- algebraic expressions can be in several different forms
  - Infix: operators are **between** operands
    - a + (b * c)
  - Prefix: operators are **before** operands
    - + a * b c
  - Postfix: operators are **after** operands
    - a b c * +
  - *Note: operands stay in same order*

algebraic expressions

- another example:
  - Infix: operators are **between** operands
    - (a + b) * c
  - Prefix: operators are **before** operands
    - * + a b c
  - Postfix: operators are **after** operands
    - a b + c *

calculator

- we would like to build a calculator which can evaluate a string containing an infix expression, e.g.
  - 2 * (3 + 4)
- we will use a two-step strategy:
  - convert string from infix to postfix
  - compute expression using a postfix calculator

calculator

- Convert this to postfix:
  - 2 * (3 + 4)

calculator

- Convert this to postfix:
  - 2 * (3 + 4)

3 4 +

calculator

- Convert this to postfix:
  - 2 * (3 + 4)

2 3 4 + *

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

stack

result operand1 operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

stack

result operand1 operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

2

stack

result operand1 operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 **3** 4 + *

2

stack

result operand1 operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 **3** 4 + *

3

2

stack

result operand1 operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 **4** + *

3

2

stack

result operand1 operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 **4** + *

4

3

2

stack

result operand1 operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

4
3
2

stack

result operand1 operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

4

3

2

stack

4

result operand1 operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

4
3
2

stack

4

result operand1 operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

3

2

stack

4

result operand1 operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

3

2

stack

4

result operand1 operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

3
2
stack

3
operand1

4
operand2

result

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

3
2
stack

3
operand1

4
operand2

result

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

2

stack

3

operand1

4

operand2

result

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

2

stack

3

operand1

4

operand2

result

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

2
stack

7
result

3
operand1

4
operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

2
stack

7
result

3
operand1

4
operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

7

2

stack

7 result

3 operand1

4 operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

7

2

stack

7 result

3 operand1

4 operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

7

2

stack

7 result

3 operand1

4 operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

7

2

stack

7 result

3 operand1

7 operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

2
stack

7
result

3
operand1

7
operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

2
stack

7
result

3
operand1

7
operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

2

stack

7

result

2

operand1

7

operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

2
stack

7
result

2
operand1

7
operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

stack

7
result

2
operand1

7
operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

stack

7
result

2
operand1

7
operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

stack

14
result

2
operand1

7
operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

stack

14
result

2
operand1

7
operand2

postfix calculator

```
// calculate a string containing postfix expression
// at loop end, the result will be at the top of the stack
for (each character ch in the string)
{
  if (ch is an operand)
    stack.push(ch)
  else // ch is an operator named op
  {
    // compute result of applying operator to the top
    // two elements and push result on stack
    operand2 = stack.peek()
    stack.pop()
    operand1 = stack.peek()
    stack.pop()
    result = operand1 op operand2
    stack.push(result)
  }
}
```

2 3 4 + *

14

stack

14

result

2

operand1

7

operand2