**Design a Calculator**

**Introduction**

I want to talk about the design of a calculator, whose operation is radically different from the calculators that most of you have used. The calculator, which we will design, implements a subset of the operations that define the Hewlett Packard 15C. The HP-15C – introduced in 1982 – is a programmable scientific calculator with rich set of scientific functions, storage for up to 67 variables, and 448 programming steps. An emulator for this classic calculator may be found [here](http://hp15c.com). To effect the design, we first need to enforce (and fill in a few gaps in) our knowledge.



**Expression Trees**

An expression tree is a binary tree in which

* Each leaf contains an operand
* The root and internal nodes are operators
* Subtrees are sub expressions, with the root of each subtree being an operator.

To prevent student overload, I will restrict my discussion to the four basic arithmetic operators (binary) { + , - , \* , / }. This in no way limits the design that we will develop.

**Example**

Suppose you have the following expression

 $\left(\left(\left(4+3\right)\*\left(5-1\right)\right) / 2\right)$

The corresponding expression tree is

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | **/** |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | \* |  |  |  |  |  | 2 |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  | + |  |  |  |  |  | - |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| 4 |  | 3 |  |  |  | 5 |  | 1 |  |  |

**Inorder Traversal**

To see that the expression tree represents the original expression, perform an inorder traversal of the above tree. Insert parenthesis around the traversal of each subtree.

 ( ( 4+3 ) \* (5-1) ) / 2)

This expression is readily processed by using the normal order of operations or by an algebraic (“normal”) calculator.

( ( 4+3 ) \* (5-1) ) / 2) evaluates to 14

Not surprisingly, this notation is called algebraic notation.

**Preorder Traversal**

What is the result of performing a preorder traversal on the same expression tree? While it is not obvious now, we will have no need of parenthesis, but I will insert extra spaces for readability.

 / \* + 4 3 - 5 1 2

Of what possible utility is this expression? To evaluate this expression, you need a different set of rules

 Repeat until a single value remains

Scan the expression from left to right until you find a triple of the form operator value value

Evaluate the triple

Replace the triple in the original expression with the value

Evaluation

/ \* + 4 3 - 5 1 2

/ \* 7 - 5 1 2

/ \* 7 4 2

/ 28 2

14

The expression resulting from a preorder traversal is called Polish Notation.

**Postorder Traversal**

Now, let’s perform a postorder traversal of the same tree. Again it is not obvious, but we will have no need of parenthesis.

4 3 + 5 1 - \* 2 /

To evaluate this expression, you need yet a different set of rules

 Repeat until a single value remains

Scan the expression from left to right until you find a triple of the form value value operator

Evaluate the triple

Replace the triple in the original expression with the value

Evaluation

4 3 + 5 1 - \* 2 /

7 5 1 - \* 2 /

7 4 \* 2 /

28 2 /

14

The expression resulting from a postorder traversal is called Reverse Polish Notation (RPN).

We will now design a calculator to process such expressions. But first a diversion to discuss an abstract data type (ADT) that we will need to design and implement the calculator.

**A Stack**

In a cafeteria, stacks of plates are stored in a spring-loaded container. Only the top plate is accessible. The container supports three operations isEmpty, addAPlate, and takeAPlate.

A stack is a data container that only allows access to the most recently added datum. A stack allows only three operations, a pop, a push, and isEmpty.

* The pop operation return the value of the most recently added datum and removes it from the stack,
* The push operation puts a new value on the stack, covering the previously most recently added value.
* The isEmpty operation returns a true if there are no items on the stack.

Sometimes a fourth operation peek is implemented; peek returns the value of the most recently added datum without removing it from the stack. Note that a peek is only a shortcut, peek = pop + push.

Consider the following depiction of a stack (labeled according to the HP conventions)

|  |  |  |
| --- | --- | --- |
| T → |  | 0 |
| Z → |  | 0 |
| Y → |  | 0 |
| X → |  | 0 |

Now let’s play with our stack

push( 2 )

|  |  |  |
| --- | --- | --- |
| T → |  | 0 |
| Z → |  | 0 |
| Y → |  | 0 |
| X → |  | 2 |

push( 6 )

|  |  |  |
| --- | --- | --- |
| T → |  | 0 |
| Z → |  | 0 |
| Y → |  | 2 |
| X → |  | 6 |

push( 121)

|  |  |  |
| --- | --- | --- |
| T → |  | 0 |
| Z → |  | 2 |
| Y → |  | 6 |
| X → |  | 121 |

a = pop( )

|  |  |  |
| --- | --- | --- |
| T → |  | 0 |
| Z → |  | 0 |
| Y → |  | 2 |
| X → |  | 6 |

**Calculator Design**

Our calculator will have a keyboard with processor, a stack, and a display.

The specification for our calculator is

* The display will display the current contents of the X position on the stack,
* Pressing keys 0…9 will cause the corresponding digit to be entered into the X position of the stack,
* Pressing the Enter key will terminate the current number entry and push the terminated entry again on the stack.
* If a key 0…9 is depressed immediately after the Enter key, the value in the X position is replace with the digit pressed.
* Pressing one of the operator keys, (/ for example) will pop two values from the stack, divide the last value popped by the first value popped and push the result on the stack.

|  |  |  |
| --- | --- | --- |
| T → |  | 0 |
| Z → |  | 0 |
| Y → |  | 2663 |
| X → |  | 663 |



**Some Examples**

To find the sum of 123 and 456, we would press the following keys

1

2

3

Enter

4

5

6

+

To find the quotient of 70/5, we would press the following keys

7

0

Enter

5

/

**Process the Expression from the Expression Tree**

Now, we are finally ready to process the expression generated by the post-order traversal of the previous expression tree

 4 3 + 5 1 - \* 2 **/**

The keystrokes to process this expression are

4

Enter

3

 +

5

Enter

1

 -

\*

2

**/**

**Bob is most definitely our uncle.**