Recursive Definition of Expressions

Expressions with binary operators can be defined as follows.

Basis: An operand is an expression.
- An operand is a variable or constant.

Induction:
1. If $E_1$ and $E_2$ are expressions, and $o$ is a binary operator (e.g., + or *), then $E_1 \circ E_2$ is an expression.
2. If $E$ is an expression, then $(E)$ is an expression.

Thus, we can build expressions like

\[
\begin{align*}
  x & \quad y & \quad z \\
  x + y & \quad (x + y) & \quad (x + y) \ast z
\end{align*}
\]

An Interesting Proof

- $S(n)$: An expression $E$ with binary operators of length $n$ has one more operand than operators.

Proof is by complete induction on the length (number of operators, operands, and parentheses) of the expression.

Basis: $n = 1$. $E$ must be a single operand. Since there are no operators, the basis holds.

Induction: Assume $S(1), S(2), \ldots, S(n)$. Let $E$ have length $n + 1 > 1$. How was $E$ constructed?

a) If by rule (2), $E = (E_1)$, and $E_1$ has length $n - 1$.

- By the inductive hypothesis $S(n-1)$, we know $E_1$ has one more operand than operators.

- But $E$ and $E_1$ have the same number of operators and operands, so $S$ holds for $E$. 
b) If by rule (1), then $E = E_1 \circ E_2$.

- Both $E_1$ and $E_2$ have length $\leq n$, because $\circ$ is one symbol and
  \[ \text{length}(E_1) + \text{length}(E_2) = n \]

- Let $E_1$ and $E_2$ have $a$ and $b$ operators, respectively. By the inductive hypothesis, which applies to both $E_1$ and $E_2$, They have $a + 1$ and $b + 1$ operands, respectively.

- Thus, $E$ has $(a + 1) + (b + 1) = a + b + 2$ operands.

- $E$ has $a + b + 1$ operators; the “$+1$” is for the $\circ$ between $E_1$ and $E_2$.

- Thus $E$ has one more operand than operator, proving the inductive hypothesis.

- Note we used all of $S(1), \ldots , S(n)$ in the inductive step.

- The fact that “expression” was defined recursively let us break expressions apart and know that we covered all the ways expressions could be built.

Recursion

- A style of programming and problem-solving where we express a solution in terms of smaller instances of itself.

- Uses basis/induction just like inductive proofs and definitions.
  - \textit{Basis} = part that requires no uses of smaller instances.
  - \textit{Induction} = solution of arbitrary instance in terms of smaller instances.

Why Recursion?

Sometimes it really helps organize your thoughts (and your code).
Example: A simple algorithm for converting integer $i > 0$ to binary: Last bit is $i \% 2$; leading bits determined by converting $i/2$ until we get down to 0.

```c
main() {
    int i;
    scanf("%d", &i);
    while(i>0) {
        putchar('0' + i%2);
        i /= 2;
    }
    putchar('
');
}
```

- Only one problem: the answer comes out backwards.
- We can fix the problem if we think recursively:

**Basis:** If $i = 0$, do nothing.

**Induction:** If $i > 0$, recursively convert $i/2$. Then print the final bit, $i \% 2$.

```c
void convert(int i) {
    if(i>0) {
        convert(i/2);
        putchar('0' + i%2);
    }
}

main() {
    int i;
    scanf("%d", &i);
    convert(i);
    putchar('
');
}
```

**Class Problem for Next Wednesday**
Prove that the above program converts its input to binary.

- What is the inductive hypothesis? The basis? The inductive step?