Expressions
Chapter 2.3 (part)
Slides #4

6 - 1\times 0 + 2 \div 2 = ?

ANSWER IT
Topics

1) How can we calculate values?
2) What's the best way to work with values like 3600?
Math Expressions

(And not like "Wow! Math is great!")
Expressions

• Expression:
  - A statement that...
  - Usually has an operator.

• Examples:
  result = 3;
  result = x * 2;
  result = 1 * x + 2;

• Expressions usable anywhere a value is needed:
  - cout << "Big number " << (1 + 2) << endl;
Order of Operations

• What is the value of result?
  ```java
  int result = 4 + 10 / 2;
  ```
  - Is it 7 or 9?   `(4 + 10) / 2`   or   `4 + (10 / 2)`

• Each operator is given a precedence:
  - Higher precedence operators are applied first.
  - `/` is higher than `+`, so the answer is..
  - Add brackets to force an ordering.

• Associatitivity:
  - Apply the operators from right-to-left, or left-to-right?
  - `+`, `-` are left to right: do the one on the..
  - `=`, `+=` are right to left: do the one on the..
Operator precedence

- Operators at same evaluated based on associativity.
  - * and / from L to R
  - = and += from R to L

- Examples:
  - result = -20 + 9 / 5;
  - result = (-20 + 9) / 5;
  - val = 6 + 5 * 4 / 3 * 2;
  - sum = sum + 10;

Order can be forced by parentheses.
See text Appendix 2 for full table.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[ ]</td>
<td>Array Index</td>
<td>L to R</td>
</tr>
<tr>
<td>2</td>
<td>+</td>
<td>unary plus</td>
<td>R to L</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>unary minus</td>
<td>R to L</td>
</tr>
<tr>
<td>3</td>
<td>* /</td>
<td>mult, div, remainder</td>
<td>L to R</td>
</tr>
<tr>
<td>4</td>
<td>+ -</td>
<td>add subtract</td>
<td>L to R</td>
</tr>
<tr>
<td>5</td>
<td>&lt;&lt; &gt;&gt;</td>
<td>stream ins. extract.</td>
<td>L to R</td>
</tr>
<tr>
<td>6</td>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>comparisons</td>
<td>L to R</td>
</tr>
<tr>
<td>7</td>
<td>= += -= * =</td>
<td>assignments</td>
<td>R to L</td>
</tr>
</tbody>
</table>
Brackets

- A statement can be correct, but unreadable:
  - `result = 1 + 2 / 6 - 1 * 3 / 4 - 3 - -3 * +4;`

- Add brackets to make it clear:
  - `result = 1 + (2 / 6) - (1 * 3 / 4) - 3 - ((-3) * (+4));`
Expression tree

- Represent \( (-6 + 5 \times 4 / (3 \times 2)) \) as a tree:

- Operands as leaves.
- Operators as branching nodes.
- Operations lower in the tree have...

- Evaluate from the..
- Write values on internal nodes.
Review

• Draw an expression tree for the following:
  answer = 5 * x + 6 * (1 – x);

  Assume x=2
Constants
Constants

- We have already used literal constants:
  ```
  int x = 10; // Numeric constant
  cout << "Hello world!\n"; // String literal
  ```

- Raw number in code are:
  ```
  int w = d / 7;
  int c = s / 72;
  ```

- Use named constants like variables:
  ```
  const int MIN_PER_HOUR = 60;
  int h = m / MIN_PER_HOUR;
  ```
const

• const qualifier makes variable...
  const int PAY_PER_DAY = 475;
  const int DAYS_PER_WEEK = 7;
  - Constants must be given a value when created.
  - Name is upper case by..
  - Program cannot modify value of a constant:
    PAY_PER_DAY = 99999; // ERROR!

• Advantages:
  - Program becomes more...
  - Can change value in entire program in one spot.
    • Ex: change tax rate that's used in 100 calculations!
Example with const

// Convert days to weeks/years/fortnights.
// #includes/uses... omitted for space.
const int DAYS_PER_WEEK = 7;
const int DAYS_PER_YEAR = 365;

int main()
{
    const int DAYS_PER_FORTNIGHT = 14;
    cout << "Enter # days: ";
    int numDays = 0;
    cin >> numDays;

    int numWeeks = numDays / DAYS_PER_WEEK;
    int numYears = numDays / DAYS_PER_YEAR;
    int numFortnight = numDays / DAYS_PER_FORTNIGHT;

    cout << "# Days:       " << setw(4) << numDays << endl;
    cout << "# Weeks:      " << setw(4) << numWeeks << endl;
    cout << "# Years:      " << setw(4) << numYears << endl;
    cout << "# Fortnights: " << setw(4) << numFortnight << endl;
}

Enter # days: 4641
# Days: 4641
# Weeks: 663
# Fortnights: 331
Guide to Constants

- Which of the following literal constants would be best made into named constants?
  - int numStudents = 0;
  - int next = numStudents + 1;
  - int waitlist = numStudents – 72;
Combined Assignments & Overflow
Assignment Operators

• Combine an operation with assignment:
  – +=, -=, *=, /=, %= 

• Examples:
  – a += b;  // means a = a + b; 
  – a *= b;  // means a = a * b; 
  – a /= 2 + 3;  // means... 

```cpp
const int MAX_COUNT = 10;
int main()
{
    int sum = 0;
    int i = 0;
    while (i < MAX_COUNT) {
        sum += i;
        i++;
    }
    cout << "Sum from 0 to " << MAX_COUNT - 1 << " = " << sum << endl;
}
```
Overflow & Underflow

• Each type has a maximum value it can store.
  − Maximum + 1 overflows to the most negative.
  − Minimum – 1 underflows to the most positive.

// Work with overflow/underflow
#include <iostream>
#include <climits>
using namespace std;
int main() {
    int test = INT_MAX;
    cout << "Test starts out at:  " << test << endl;
    test += 1;
    cout << "Adding one gives us:  " << test << endl;
    test -= 1;
    cout << "Now subtracting 1:   " << test << endl;
}

#include <climits>..

INT_MAX, INT_MIN,
CHAR_MAX, CHAR_MIN,
....
Suggested Review Questions

• Draw an expression tree for the following:  
  \[ \text{result} = 8 \times -1 + 3 / 2 \]
  - Solve it by writing values on the nodes.
  - Write a C++ program to double check your answer.
Summary

- Expressions calculate values using operators.
  - Operator precedence gives us expression trees.
- Use named constants (const), not magic numbers.
- Combined assignment operators like `x += 2;`