

CSC 221: Computer Programming I

Fall 2001

- expressions revisited
 - arithmetic operators, precedence & associativity
- assignments
 - memory cells, evaluation order, declaration+initialization
- math functions
 - cmath library, pow, sqrt, fabs, ...
- constants
- formatted output
 - io manip library, setiosflags, setprecision, setw

arithmetic expressions

recall from last week: once you have values stored in variables, can apply arithmetic operators to form *expressions*

+ <i>addition</i>	- <i>subtraction</i>	
* <i>multiplication</i>	/ <i>division</i>	% <i>remainder</i>
1 + 23	→ 24	
2 * 2 * 2	→ 8	can chain operators together in an expression
3 + 2.5 + 8.0	→ 13.5	can mix integers and reals: integer values are first <i>coerced</i> into reals, then the operator is applied
1 + 2 * 3	→ 7	* and / have higher <i>precedence</i> than + and -
(1 + 2) * 3	→ 9	can force evaluation order with parentheses
8 / 4 / 2	→ 1	similar operators are evaluated left-to-right (i.e., have left <i>associativity</i>)

don't rely on precedence/associativity rules – parenthesize to make order clear

avoiding integer division

if possible, use a real value in the numerator and/or denominator

- will force real division, e.g.,

```
(5.0/9.0 * (tempInFahr - 32))
(5.0/9 * (tempInFahr - 32))
(5/9.0 * (tempInFahr - 32))
```

when values are stored in variables, not quite so simple

- can use type name to *coerce* from one type to another

```
double(5) → 5.0           int(5.2) → 5
```

```
int numerator, denominator;
. . .

(double(numerator)/denominator * (tempInFahr - 32))
```

assignment statements

recall: a variable is a piece of memory where a value can be stored

- so far, have assigned value via cin statement
used to store user input
- can also assign value explicitly using an assignment statement
used to store values or intermediate computations within the program

```
variableName = valueOrExpression;
```

```
age = 21;           21
                   age

firstName = "Dave"; "Dave"
                   firstName

tempInCelsius = (5.0/9.0 * (41 - 32)); 5.0
                                       tempInCelsius
```

order of evaluation

when an assignment statement is executed:

1. evaluate the expression on the right-hand side
2. assign the resulting value to the variable on the left-hand side

```
product = 2 * 3 * 4 * 5; 120  
product
```

```
product = product + 1; 121  
product
```

danger: a variable has an *undefined* value until explicitly assigned

- some bit pattern remains in memory, no way of knowing what

```
int sum; ???  
sum
```

```
sum = sum + 1; ???+1  
sum
```

in-class exercise: trace the following

```
int num1, num2, num3;  
double average;  
  
num1 = 0;  
  
num2 = 5;  
  
average = (num1 + num2)/2.0;  
  
num1 = num1 + 2*num2;  
  
num2 = num1/2;  
  
num3 = 9;  
  
num3 = num3 % 2;  
  
average = num2 * 1.5;
```

declaration + initialization

to reduce the risk of uninitialized variables, can declare and initialize variables at the same time

```
variableType variableName = valueOrExpression;

int sum = 0;

double tempInCelsius = (5.0/9.0 * (tempInFahr - 32));
```

preferred style: wait and declare/initialize variables when needed

- makes program development more fluid
- when decide you need a variable, declare & initialize it there & keep going

assignment example

- number of each coin type is assigned to a new variable

are there alternative ways to update the amount?

- the '\t' character yield a TAB when printed

- what if we wanted to allow for 50¢ pieces?

```
// change.cpp      Dave Reed      9/6/01
////////////////////////////////////

#include <iostream>
using namespace std;

int main()
{
    int amount;
    cout << "Enter an amount (in cents) to make change: ";
    cin >> amount;

    int quarters = amount/25;
    amount = amount%25;

    int dimes = amount/10;
    amount = amount%10;

    int nickels = amount/5;
    amount = amount%5;

    int pennies = amount;

    cout << "Optimal change:" << endl;
    cout << "   # of quarters =\t" << quarters << endl;
    cout << "   # of dimes =\t" << dimes << endl;
    cout << "   # of nickels =\t" << nickels << endl;
    cout << "   # of pennies =\t" << pennies << endl;

    return 0;
}
```

math functions

the `cmath` library contains useful mathematical functions

- a *function* is a mapping from some number of inputs to a single output
- in C++, a function is called by name, with its inputs in parentheses
the output of the function replaces the call in an expressions

```
#include <cmath>

num = fabs(-5.3);           // num = |-5.3| = 5.3
x = sqrt(9.0);             // x =  $\sqrt{9.0}$  = 3.0
k = pow(2.0, 10.0);        // k =  $2^{10}$  = 1,024
down = floor(2.4);         // down =  $\lfloor 2.4 \rfloor$  = 2.0
up = ceil(2.4);            // up =  $\lceil 2.4 \rceil$  = 3.0
```

math example

- determines the distance between two points using the `pow` and `sqrt` functions

```
// distance.cpp          Dave Reed          9/6/01
//
// This program determines the distance between two points.
//
//
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
    double x1, y1, x2, y2;

    cout << "Enter the x and y coordinates of the 1st point: ";
    cin >> x1 >> y1;
    cout << "Enter the x and y coordinates of the 2nd point: ";
    cin >> x2 >> y2;

    double distance = sqrt(pow(x1-x2, 2.0) + pow(y1-y2, 2.0));

    cout << endl << "The distance between (" << x1 << ", "
        << y1 << ") and (" << x2 << ", " << y2 << ") is "
        << distance << endl;

    return 0;
}
```

another example

technically, the functions in `cmath` take doubles as inputs, return a double as output

- ok to provide ints as inputs – automatically coerced into double
- ok to assign return value to int variable, similarly coerced

```
int(pow(2, numBits))
```

```
// bits.cpp      Dave Reed      9/6/01
//
// This program determines the distance between two points.
///////////////////////////////////////////////////////////////////

#include <iostream>
#include <cmath>
using namespace std;

int main()
{
    int numBits;
    cout << "Enter a number of bits: ";
    cin >> numBits;

    int numValues = pow(2, numBits);

    cout << endl << "With " << numBits
         << " bits, you can represent "
         << numValues << " values." << endl;

    return 0;
}
```

FYI: 1 KB = 2^{10} bytes 1 MB = 2^{20} bytes 1 GB = 2^{30} bytes

puzzler...

can we determine the range of values for the different integer types?
recall: different types provide different ranges of values, can vary by machine

- with Visual C++, `short` is stored using 16 bits $\rightarrow 2^{16}$ values
 $2^{16} = 65,536$ different values, but must be divided among neg., 0, & pos.
 $-2^{15} \dots 0 \dots (2^{15}-1) \rightarrow -32,768 \dots 0 \dots 32,767$
- with Visual C++, `int` and `long` are both stored using 32 bits $\rightarrow 2^{32}$ values
by similar reasoning, range is $-2^{31} \dots 0 \dots (2^{31}-1)$
but 2^{31} is too big to fit in an `int` or `long` value, so can't compute it!
could compute 2^{30} and double by hand
or, could compute -2^{31} by changing first argument in `pow` to be -2
 $-2,147,483,648 \dots 0 \dots 2,147,483,647$

useful example

- note the use of (long) meaningful variable names – makes program easier to read/debug
- values are read in and stored as doubles – even if integer values are entered by user
- note "magic number" 3.14159 – what if we wanted more precision?

```
// pizza.cpp      Dave Reed      9/6/01
//
// This program determines the cost per sq. inch of a pizza.
///////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;

int main()
{
    double pizzaDiameter, pizzaCost;
    cout << "Enter the diameter of the pizza (in inches): ";
    cin >> pizzaDiameter;
    cout << "Enter the price of the pizza (in dollars): ";
    cin >> pizzaCost;

    double pizzaArea = 3.14159*(pizzaDiameter*pizzaDiameter)/4.0;
    double costPerSqInch = pizzaCost/pizzaArea;

    cout << "Total area of the pizza: " << pizzaArea
         << " square inches." << endl;
    cout << "    price per square inch = $"
         << costPerSqInch << "." << endl;

    return 0;
}
```

constants

if a variable is to be assigned a value once & never changed, declare it to be a *constant*

- precede variable declaration/assignment with keyword `const`

```
const double PI = 3.14159;
```

advantages:

1. safety: the compiler will catch any attempt to change the constant, yielding an error message upon compilation
2. readability: since the value will never change, you can place all constant definitions at the top of the program
→ easy to find & modify

constant example

if we wanted more digits of precision for PI, just look at the top & change

also, use of constant names in code makes expressions more readable

note style convention: I use all CAPS in constant names to distinguish from standard variables

```
// pizza.cpp      Dave Reed      9/6/01
//
// This program determines the cost per sq. inch of a pizza.
///////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;

const double PI = 3.14159;

int main()
{
    double pizzaDiameter, pizzaCost;
    cout << "Enter the diameter of the pizza (in inches): ";
    cin >> pizzaDiameter;
    cout << "Enter the price of the pizza (in dollars): ";
    cin >> pizzaCost;

    double pizzaArea = PI*(pizzaDiameter*pizzaDiameter)/4.0;
    double costPerSqInch = pizzaCost/pizzaArea;

    cout << "Total area of the pizza: "
         << pizzaArea << " square inches." << endl;
    cout << "    price per square inch = $"
         << costPerSqInch << "." << endl;

    return 0;
}
```

constant example (cont.)

suppose we also wanted to know the circumference of the pizza (amount of outer crust)

- this computation also uses the constant PI
- having a single constant ensures that the formulas are consistent

```
// pizza.cpp      Dave Reed      9/6/01
//
// This program determines the cost per sq. inch of a pizza.
///////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;

const double PI = 3.14159;

int main()
{
    double pizzaDiameter, pizzaCost;
    cout << "Enter the diameter of the pizza (in inches): ";
    cin >> pizzaDiameter;
    cout << "Enter the price of the pizza (in dollars): ";
    cin >> pizzaCost;

    double pizzaArea = PI*(pizzaDiameter*pizzaDiameter)/4.0;
    double costPerSqInch = pizzaCost/pizzaArea;

    double pizzaCircumference = PI*pizzaDiameter;

    cout << "Total area of the pizza: "
         << pizzaArea << " square inches." << endl;
    cout << "    price per square inch = $"
         << costPerSqInch << "." << endl;
    cout << "Outer crust: " << pizzaCircumference
         << " inches." << endl;

    return 0;
}
```

formatting output

by default, very large or very small real values will be printed in scientific notation

$1.234e9 \rightarrow 1.234 \times 10^9 \rightarrow 1,234,000,000$

$1.234e-9 \rightarrow 1.234 \times 10^{-9} \rightarrow 0.000000001234$

reasonably sized reals will be printed with (up to) 6 digits to the right of the decimal place

- can override these defaults with routines from the `iomanip` library

```
#include <iomanip>
```

- need to send a format manipulator to `cout` to specify fixed-width decimals

```
cout << setiosflags(ios::fixed);
```

- need to send a format manipulator to specify the precision (number of digits to the right of the decimal place) as desired

```
cout << setprecision(2) << costPerSqInch << endl;
```

format example

note: only need to call `setiosflags` once

each call to `setprecision` sets the default precision, so can change repeatedly

```
// pizza.cpp      Dave Reed      9/6/01
//
// This program determines the cost per sq. inch of a pizza.
///////////////////////////////////////////////////////////////////

#include <iostream>
#include <iomanip>
using namespace std;

const double PI = 3.14159;

int main()
{
    double pizzaDiameter, pizzaCost;
    cout << "Enter the diameter of the pizza (in inches): ";
    cin >> pizzaDiameter;
    cout << "Enter the price of the pizza (in dollars): ";
    cin >> pizzaCost;

    double pizzaArea = PI*(pizzaDiameter*pizzaDiameter)/4.0;
    double costPerSqInch = pizzaCost/pizzaArea;

    cout << setiosflags(ios::fixed);

    cout << "Total area of the pizza: "
         << setprecision(4) << pizzaArea << " square inches."
         << endl;
    cout << "    price per square inch = $"
         << setprecision(2) << costPerSqInch << "." << endl;

    return 0;
}
```

more formatting

- the `setw` manipulator sets the column width of the output (right-justified)

unlike `setprecision`, `setw` only affects the format of the next expression printed

```
// change.cpp          Dave Reed          9/6/01
////////////////////

#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    int amount;
    cout << "Enter an amount (in cents) to make change: ";
    cin >> amount;

    int quarters = amount/25;
    amount = amount%25;

    int dimes = amount/10;
    amount = amount%10;

    int nickels = amount/5;
    amount = amount%5;

    int pennies = amount;

    cout << "Optimal change:" << endl;
    cout << setw(5) << quarters << " quarters" << endl;
    cout << setw(5) << dimes << " dimes" << endl;
    cout << setw(5) << nickels << " nickels" << endl;
    cout << setw(5) << pennies << " pennies" << endl;

    return 0;
}
```

in-class example:

consider computing income tax using a flat-tax rate:

- read in income, deductions, and amount withheld
- compute taxable income, total tax, tax owed
- display results

what do we need?

- read in income, deductions, and amount withheld
 - variables to store the three values: `income`, `deductions`, `withheld`
 - `cout` to prompt user, `cin` to read in values
- compute taxable income, total tax, tax owed
 - constants for tax rate (13%) and standard deduction (\$4,400)
 - variables for computed values
 - taxable income = income – deductions
 - total tax = taxable income * (tax rate/100.0)
 - tax owed = total tax – withheld
- display results
 - `cout` statements to display, `setprecision` to format

tax example

```
// flattax.cpp      Dave Reed      9/6/01
////////////////////

#include <iostream>
#include <iomanip>
using namespace std;

const double TAX_RATE = 13.0;
const double STANDARD_DEDUCTION = 4400;

int main()
{
    double income, deductions, withheld;
    cout << "Gross income: ";
    cin >> income;
    cout << "Deductions (" << STANDARD_DEDUCTION << " if not itemizing): ";
    cin >> deductions;
    cout << "Amount already withheld: ";
    cin >> withheld;

    double taxableIncome = income - deductions;
    double totalTax = taxableIncome * TAX_RATE/100.0;
    double taxOwed = totalTax - withheld;

    cout << setiosflags(ios::fixed) << setprecision(2);

    cout << endl;
    cout << "Your taxable income is $" << taxableIncome << endl
         << "Your total tax liability is $" << totalTax << endl
         << "  which means you still owe $" << taxOwed << endl;

    return 0;
}
```