ASSIGNMENT

Review

- The C++ assignment operator is:

  =

  - Do NOT confuse assignment with math equality!

    $(x + 1)^2 = x^2 + 2x + 1$

- Remember our syntax rule:
  - The LHS must be one variable
  - The RHS is an ***expression*** that evaluates to a ***compatible data type*** as the variable

- Example:
  - `y = x - 7;  // variable = expression`

- Syntax error:
  - `x - 7 = y;  // syntax error`

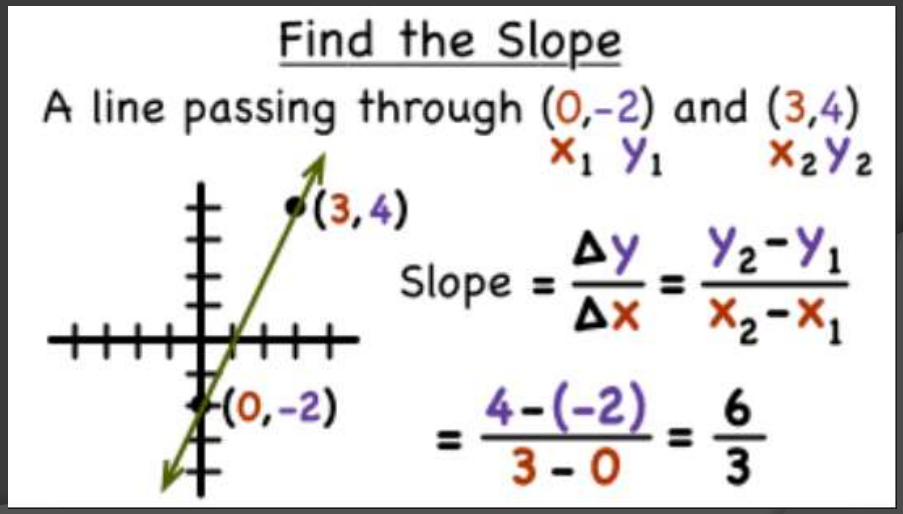
Expressions

- An *expression* is a formula
  - Values: Constants, variables, and function calls
  - Operators: +, -, *, /, etc
  - Parenthesis: grouping
- Operators combine values to evaluate to a single result
- The RHS is evaluated to a single value and then it is assigned the variable on the LHS
- Examples:

```
double x = 3.0 * 4.0;
double y = 2.0 + x;
double z = 5.0 + x / y - sqrt(x * 3.0);
double a = (2.5 + 1.0) * 3.0;
```

Write a Program: Line Slope

- Write a C++ program to compute the slope of a line in Cartesian coordinates, i.e. the x-y plane given two points



Steps to Develop Your Algorithm

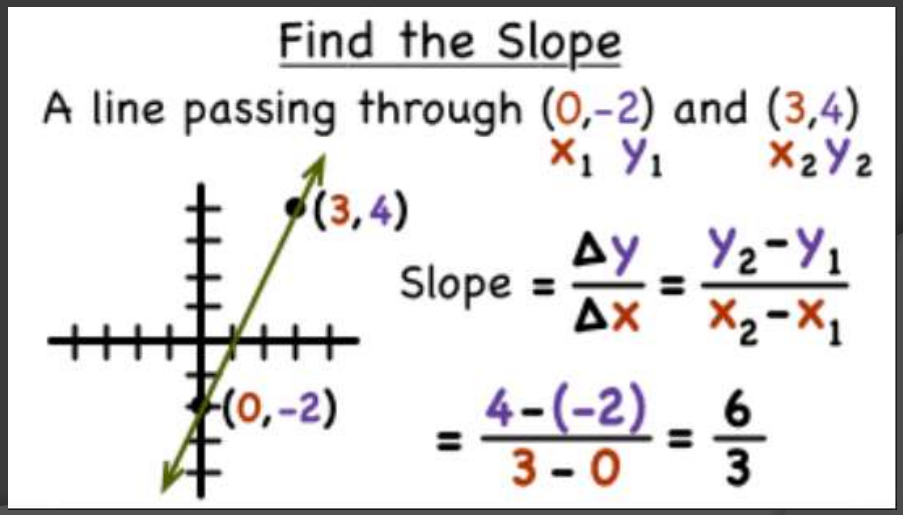
- Write a C++ program to compute the slope of a line in Cartesian coordinates, i.e. the x-y plane given two points

- Step 1: What values do we need to input from the user?

Answer:

$(x_1, y_1) = (0, -2)$

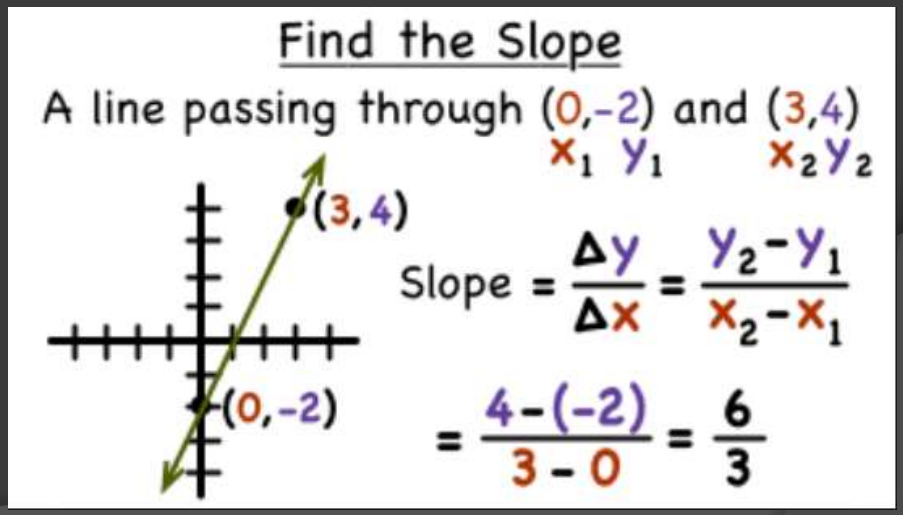
$(x_2, y_2) = (3, 4)$



Steps to Develop Your Algorithm

- Write a C++ program to compute the slope of a line in Cartesian coordinates, i.e. the x-y plane given two points
- Step 1a: Write code to read in user input

```
cout << "Enter …"
cin >> x1 >> y1;
cin >> x2 >> y2;
```



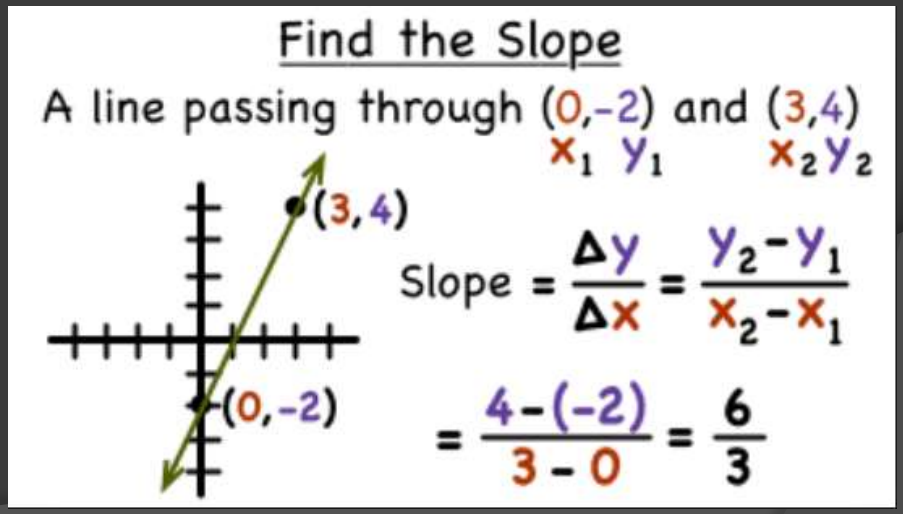
Steps to Develop Your Algorithm

- Write a C++ program to compute the slope of a line in Cartesian coordinates, i.e. the x-y plane given two points
- Step 2: What values do you need to calculate?

Answer:

Delta-Y (y-diff)

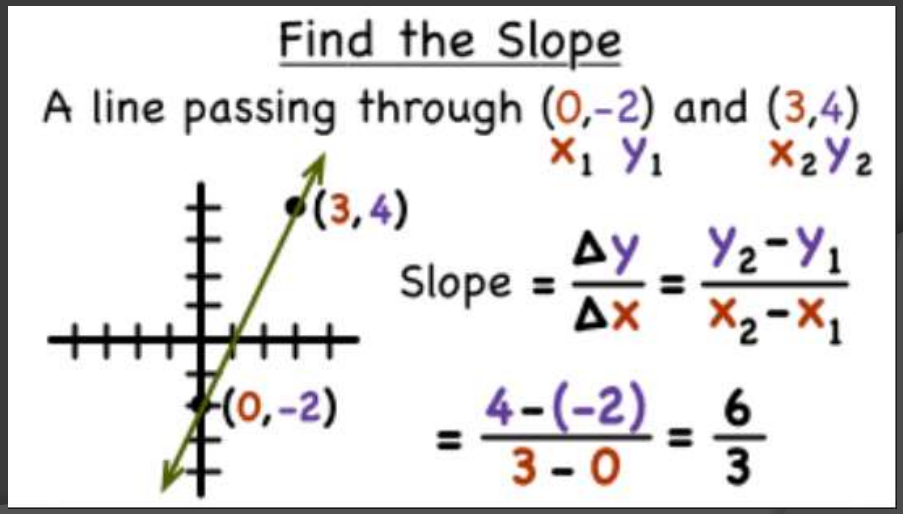
Delta-X (x-diff)



Steps to Develop Your Algorithm

- Write a C++ program to compute the slope of a line in Cartesian coordinates, i.e. the x-y plane given two points
- Step 2a: Write code to calculate …

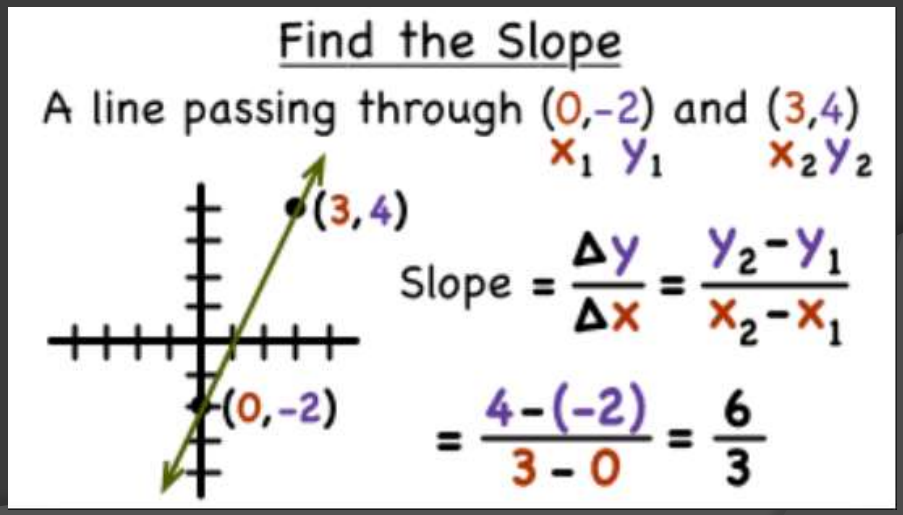
```
xdiff = x1 - x2;
ydiff = y1 - y2;
slope = ydiff /
   xdiff;
```



Steps to Develop Your Algorithm

- Write a C++ program to compute the slope of a line in Cartesian coordinates, i.e. the x-y plane given two points
- Step 3: Output slope

```
cout >>
     slope >> endl;
```



simpleslope.cpp

```
// Compute the slope between two points
#include <iostream>
using namespace std;

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

  cout << "Enter x and y coordinates of first point : ";
  cin >> x1 >> y1;
  cout << "Enter x and y coordinates of second point : ";
  cin >> x2 >> y2;

  xdiff = x1 - x2;
  ydiff = y1 - y2;
  slope = ydiff / xdiff;
  cout << "The slope is: " << slope << endl;

  return 0;
}
```

Your Turn: BMI

- Write a C++ program to compute a person's body mass index, which is computed as 705 times their weight divided by their height squared
- What values do we need to prompt and read from the user?
- What will the expression to calculate bmi look like?

bmi.cpp

OPERATOR PRECEDENCE

Operator Precedence

- What does this expression evaluate to?

```
1 + 3 * 6 - 4 / 2 = ???
```

Try it out on a sheet of paper

Operator Precedence

```
1 + 3 * 6 - 4 / 2
```

- Multiplication and division are done BEFORE addition and subtraction
  - * and / have higher precedence than + and -
- Lets use grouping, i.e. parenthesis, to show which operations are done first

```
((1 + (3 * 6)) - (4 / 2))
```

Operator Precedence and Associativity

- Operators with higher operator precedence are evaluated first
- If operators have the same precedence, the evaluate from left to right

Precedence	**Associativity**
() All Unary Operators Example: Unary -	
* / %	Left to right
+ -	Left to right

Minus Sign

- The minus sign is an *overloaded* operator
- Binary operator
  - Subtraction
  - `int a = b - c;`
- Unary operator
  - Negation
  - `int a = -c;`
- Depends on the context, operator precedence, and associativity rules

Your Turn: arithmetic3.cpp

```
// Precedence of arithmetic operators

#include <iostream>
using namespace std;

int main()
{
  cout << "-3+5*2 = " << -3+5*2 << endl << endl;

  // Is this?
  cout << "((-3)+5)*2 = " << ((-3)+5)*2 << endl;
  cout << "(-(3+5))*2 = " << (-(3+5))*2 << endl;
  cout << "(-3)+(5*2) = " << (-3)+(5*2) << endl;
  cout << "-(3+(5*2)) = " << -(3+(5*2)) << endl;

  return 0;
}
```

Another Example: power.cpp

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

int main()
{
  double x, y;
  cout << "Enter x : ";
  cin >> x;

  y = x;
  y = y * y;        // Note: computes y * y then assigns to y.
  cout << "x^2 = " << y << endl;

  y = y * x;        // Note: computes y * x then assigns to y.
  cout << "x^3 = " << y << endl;

  y = y * x;        // Note: computes y * x then assigns to y.
  cout << "x^4 = " << y << endl;

  return 0;
}
```

Accumulation Variables

```
y = x;
y = y * y;
y = y * x;
y = y * x;
```

- The variable y is called an *accumulation variable* because it is temporarily holding a partial solution over many steps

Note a Common ERROR!

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

int main()
{
  double x, y;

  cout << "Enter x : ";
  cin >> x;

  y = x^2;        // SYNTAX ERROR.
  cout << "x^2 = " << y << endl;

  y = x^3;        // SYNTAX ERROR.
  cout << "x^3 = " << y << endl;

  y = x^4;        // SYNTAX ERROR.
  cout << "x^4 = " << y << endl;

  return 0;
}
```

```
…
13.     y = x^2;          // SYNTAX ERROR.
14.     cout << "x^2 = " << y << endl;
15.
16.     y = x^3;          // SYNTAX ERROR.
17.     cout << "x^3 = " << y << endl;
18.
19.     y = x^4;          // SYNTAX ERROR.
20.     cout << "x^4 = " << y << endl;
…
```

```
> g++ powerError.cpp
powerError.cpp: In function `int main()':
powerError.cpp:13: invalid operands of types `double' and `int' to binary `operator^'
powerError.cpp:16: invalid operands of types `double' and `int' to binary `operator^'
powerError.cpp:19: invalid operands of types `double' and `int' to binary `operator^'
>
```

ASSIGNMENT VARIATIONS

Useful shortcuts

- The statement:

```
counter = counter + 5;
```

can also be written as

```
counter += 5
```

- General syntax: `variable += expression;`
  - Evaluates the expression
  - Adds it to the current value of the variable
  - Assigns this answer to the variable

- Examples:

```
sum += 6;
sum += dogs + cats;
```

- The last example is shorthand for:

```
sum = sum + (dogs + cats);
```

Useful Shortcuts

- Memorize these similar shortcuts: +=, -=, *=, /=, and %=

```
product *= ratio;
```

- Your Turn:

```
double product = 10.0, ratio = 0.5;
product *= ratio + 1;
```

- What is the final value of `product`?

Useful Shortcuts

- Your Turn:

```
double product = 10.0, ratio = 0.5;
product *= ratio + 1;
```

- What is the final value of `product`?

```
product = product * (ratio + 1);
product = product * (0.5 + 1);
product = product * 1.5;
product = 10 * 1.5;
product = 15;
```

Remember: Accumulation Variables

- Trace through this code and track the value of the variable `total`

```
int total = 0;
total += 6;      //total is now 6
total += 3;      //total is now 9
total += 8;      //total is now 17
```

- These types of statements are very useful!

Start using them!

Useful Shortcuts: Increment/Decrement

- A lot of programmers like to update their variable values by 1, e.g. counting how many
- The increment operator ++ is preferred by programmers

```
i = i + 1;
i += 1;
i++;          // this is post increment
```

- There is also the decrement operator --

```
i = i - 1;
i -= 1;
i--;          // this is post decrement
```

Pre-increment vs post-increment

- Consider:

```
int i = 1;
cout << i++ << endl;
cout << i;
```

- Outputs:

```
1
2
```

- But …

```
int i = 1;
cout << ++i << endl; // Pre-increment operator
cout << i;
```

- Outputs:

```
2
2
```

But remember ...

- ++ and -- are *unary* operators
  - They have higher precedence than other operators
  - Even if they are *post* increment/decrement
- To illustrate:

```
int x(5);
cout << x-- << x << endl;
```

- Output:

```
5 4
```

- Just remember: all unary operators have highest precedence

VARIABLE INITIALIZATION

Shorthand

- Combine variable declaration and initialization

```
double x = 2.5;
```

- Instead

```
double x(2.5);
```

- General form

  data type variable (expression);

initExample2.cpp

```
// initialization example

#include <iostream>
#include <cmath>

using namespace std;

int main()
{
  double x(3.0 + 4.0);          // initialize x to 3 + 4
  double y(5.0), z(6.0);        // initialize y and z

  cout << "The reciprocal of 3 + 4 is " << 1 / x << endl;
  cout << "The reciprocal of " << y << " is " << 1 / y << endl;
  cout << "The reciprocal of " << z << " is " << 1 / z << endl;

  return 0;      // exit program
}
```

initExample3.cpp

```
// C style initialization example

#include <iostream>
#include <cmath>

using namespace std;

int main()
{
  double x = 3.0 + 4.0;   // C style initialization of x to 3 + 4
  double y = 5.0, z = 6.0; // initialize y and z

  cout << "The reciprocal of 3+4 is " << 1 / x << endl;
  cout << "The reciprocal of " << y << " is " << 1 / y << endl;
  cout << "The reciprocal of " << z << " is " << 1 / z << endl;

  return 0;      // exit program
}
```

noInit1.cpp

```
// example of missing initialization

#include <iostream>
#include <cmath>

using namespace std;

int main()
{
  double x;
  double y(123.456);

  cout << "x = " << x << endl;
  cout << "y = " << y << endl;
  cout << "e^(" << x << ") = " << exp(x) << endl;
  cout << "x + y = " << x + y << endl;

  return 0;      // exit program
}
```

noInit1.cpp

```
…
   double x;
   double y(123.456);

   cout << "x = " << x << endl;
   cout << "y = " << y << endl;
   cout << "e^(" << x << ") = " << exp(x) << endl;
   cout << "x + y = " << x + y << endl;
…
```

```
> noInit1.exe
x = 0
y = 123.456
e^(0) = 1
x+y = 123.456

>
```

noInit2.cpp – rearrange declarations

```
// example of missing initialization

#include <iostream>
#include <cmath>

using namespace std;

int main()
{
  double y(123.456);
  double x;

  cout << "x = " << x << endl;
  cout << "y = " << y << endl;
  cout << "e^(" << x << ") = " << exp(x) << endl;
  cout << "x + y = " << x + y << endl;

  return 0;      // exit program
}
```

noInit2.cpp

```
…
   double y(123.456);
   double x;

   cout << "x = " << x << endl;
   cout << "y = " << y << endl;
   cout << "e^(" << x << ") = " << exp(x) << endl;
   cout << "x + y = " << x + y << endl;
…
```

```
> noInit2.exe
x = -7.69536e+304
y = 123.456
e^(-7.69536e+304) = 0
x+y = -7.69536e+304

>
```

Forgetting to Declare a Variable

```
1.  // Examples of an undeclared variable
2.
3.  #include <iostream>
4.  #include <cmath>
5.  using namespace std;
6.
7.  int main()
8.  {
9.    double y(0.0);
10.   y = 2.0 + x;
11.
12.   double x(0.0);
13.   x = 3.0 * 4.0;
14.
15.   cout << "x = " << x << endl;
16.   cout << "y = " << y << endl;
17.
18.   return 0;
19. }
```

```
…
7.    int main()
8.    {
9.      double y(0.0);
10.     y = 2.0 + x;
11.
12.     double x(0.0);
13.     x = 3.0 * 4.0;
14.
15.     cout << "x = " << x << endl;
16.     cout << "y = " << y << endl;
…
```

```
> g++ undeclaredVariable.cpp –o undeclaredVariable.exe
undeclaredVariable.cpp: In function `int main()':
undeclaredVariable.cpp:10: `x' undeclared (first use this function)
undeclaredVariable.cpp:10: (Each undeclared identifier is reported only once
   for each function it appears in.)
>
```

Warning

Remember: Make sure you have already assigned values to variables BEFORE they are used in these computations!

COERCION

Coercion

- Assigning an integer to a floating point variable ***converts*** the integer to a floating point number

  Example:

```
double y = 3;
```

- Assigning a floating point number to an integer ***truncates*** the number and ***converts*** it to an integer

  Example:

```
int x = 3.4;
```

coercion.cpp

```
1.  // assigning float to integer or integer to float
2.
3.  #include <iostream>
4.  using namespace std;
5.
6.  int main()
7.  {
8.    int x(0);
9.    double y(0.0);
10.
11.   x = 3.4;    // assign floating point number to int
12.   y = 3;      // assign integer to a floating point variable
13.
14.   cout << "x = " << x << endl; // x equals 3
15.   cout << "y = " << y << endl; // 3 is output, but y is still a float
16.
17.      cout << "1/x = " << 1/x << endl;  // 0 since x is an integer
18.      cout << "1/y = " << 1/y << endl;  // 0.333333 since y is a float
19.
20.      return 0;
21. }
```

```
…
8.    int x(0);
9.    double y(0.0);
10.
11.   x = 3.4;              // assign floating point number to int
12.   y = 3;                // assign integer to a floating point variable
13.
14.   cout << "x = " << x << endl; // x equals 3.
15.   cout << "y = " << y << endl; // 3 is output.. but y is still a float
16.
17.   cout << "1/x = " << 1/x << endl;      // 0 since x is an integer
18.   cout << "1/y = " << 1/y << endl;      // 0.333333 since y is a float
…
```

```
> g++ coercion.cpp –o coercion.exe
coercion.cpp: In function `int main()':
coercion.cpp:11: warning: assignment to `int' from `double'
coercion.cpp:11: warning: argument to `int' from `double'
>
```

```
…
8.     int x(0);
9.     double y(0.0);
10.
11.     x = 3.4;     // assign floating point number to int
12.     y = 3;       // assign integer to a floating point variable
13.
14.     cout << "x = " << x << endl;     // x equals 3.
15.     cout << "y = " << y << endl;     // 3 is output.. but y is still a float
16.
17.      cout << "1 / x = " << 1 / x << endl;     // 0 since x is an integer
18.       cout << "1 / y = " << 1 / y << endl;    // 0.333333 since y is a float
…
```

```
> coercion.exe
x = 3
y = 3
1/x = 0
1/y = 0.333333
>
```

Coercion

- Remember the data type ***character***?
  - A character is represented as an integer with size of a single byte
- Assigning an integer to a char variable converts the integer to a character
- Example:

```
char c = 88;        // c is now 'X'
                    // See your ASCII table
```

coercion2.cpp

```
#include <iostream>
using namespace std;
int main()
{
  int x1(71), x2(111), x3(32), x4(66), x5(117), x6(99), x7(107), x8(115),
    x9(33);
  char c1, c2, c3, c4, c5, c6, c7, c8, c9;

  c1 = x1;
  c2 = x2;
  c3 = x3;
  c4 = x4;
  c5 = x5;
  c6 = x6;
  c7 = x7;
  c8 = x8;
  c9 = x9;

  cout << c1 << c2 << c3 << c4 << c5 << c6 << c7 << c8 << c9 << endl;

  return 0;
}
```

What is the output?