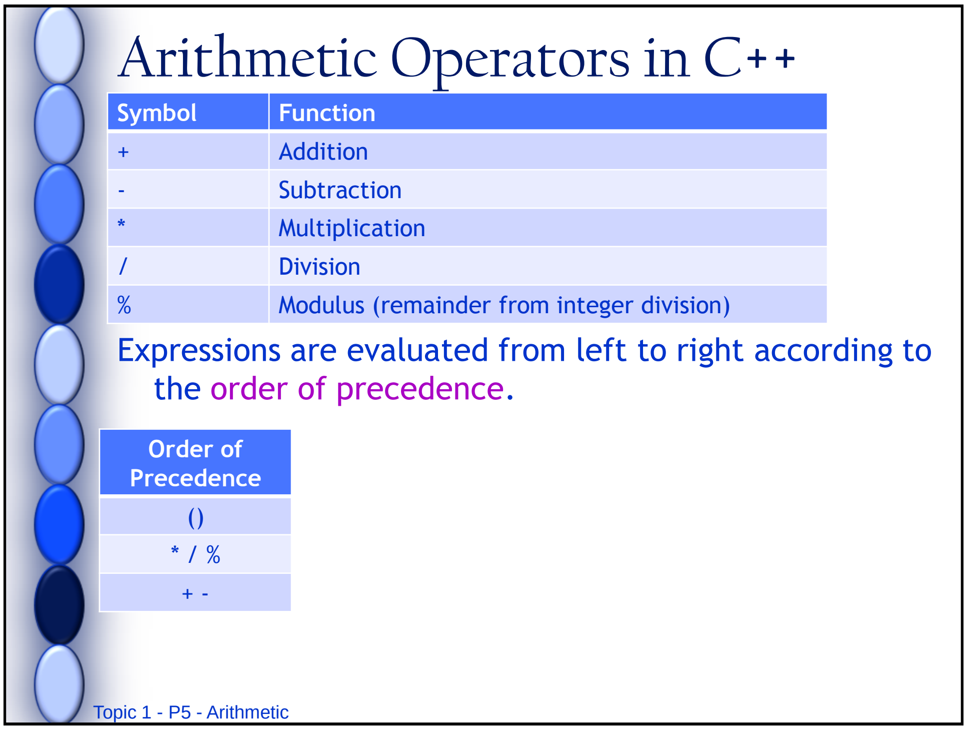
Topic 1 – CS1A Review – P5 - Arithmetic in C++

Chapter 4 in the shrinkwrap

Part 1

Announcement

- Lab 2 - Arithmetic

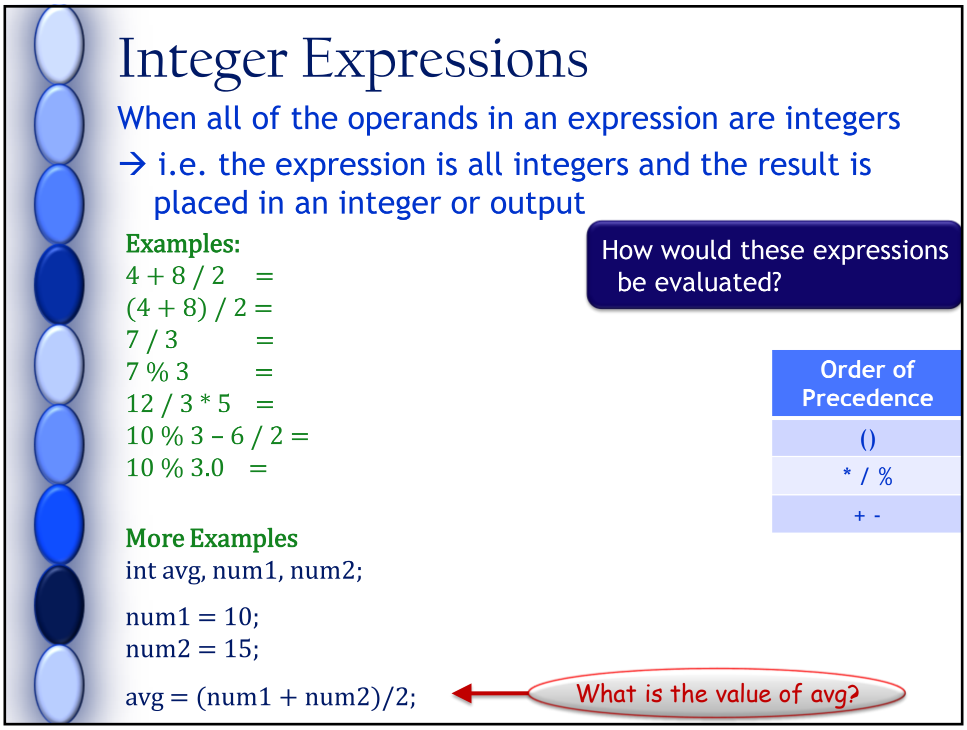


Arithmetic Operators in C++

Symbol	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (remainder from integer division)

Expressions are evaluated from left to right according to the order of precedence.

Order of Precedence
()
* / %
+ -



Integer Expressions

When all of the operands in an expression are integers

→ i.e. the expression is all integers and the result is placed in an integer or output

Examples:

```
4 + 8 / 2     =
(4 + 8) / 2 =
7 / 3         =
7 % 3         =
12 / 3 * 5    =
10 % 3 – 6 / 2 =
10 % 3.0      =
```

How would these expressions be evaluated?

Order of Precedence
()
* / %
+ -

More Examples

```
int avg, num1, num2;

num1 = 10;
num2 = 15;

avg = (num1 + num2)/2;
```

What is the value of avg?

Floating Point Expressions

- When all the operands are floats and the result is placed in a float or output

Examples:

```
5.0 * 2.0 / 4.0 * 3.0     =
5.0 * 2.0 / (4.0 * 2.0)  =
5.0 + 2.0 / (4.0 * 2.0) =
```

More Examples

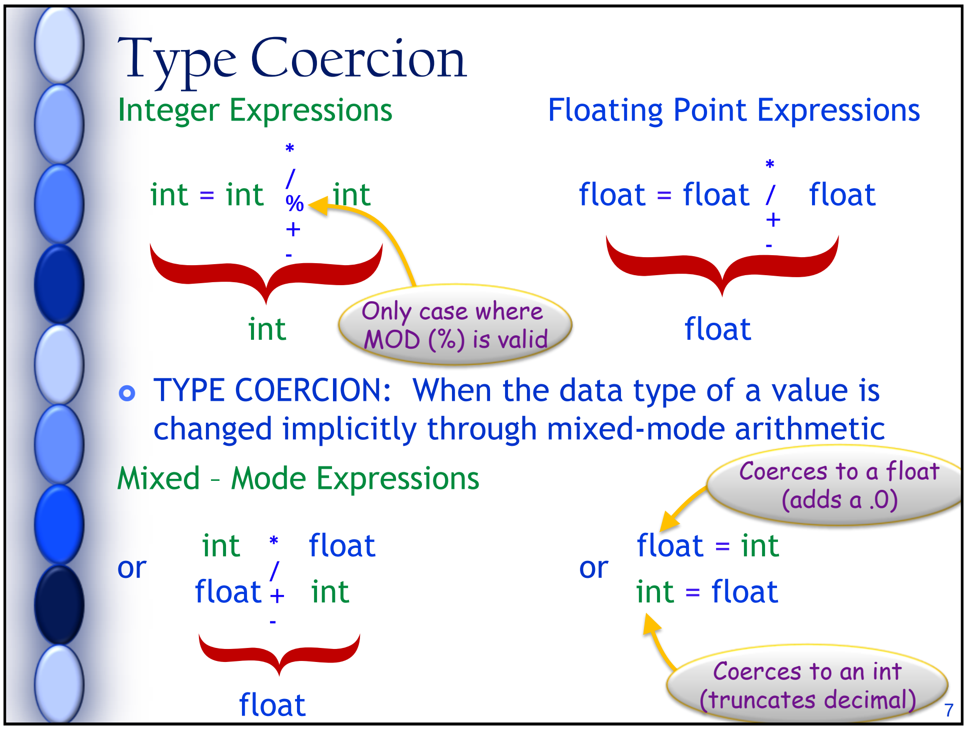
```
float  sum, num1, num2;

num1 = 10.0;
num2 = 15.5;

avg = (num1 + num2)/2.0;
```

What is the value of avg?

Mixed Mode Arithmetic

- Two types of data types to represent numeric values
  - int & float
  - They store data differently
  - Allocate memory differently
    - i.e. int 6 is stored differently than float 6.0

- Mixed mode arithmetic
  - → when we combine different data types
  - e.g. float & int



Type Coercion

Integer Expressions

$$\underbrace{\text{int} = \text{int} \begin{matrix} * \\ / \\ \% \\ + \\ - \end{matrix} \text{int}}_{\text{int}}$$

Only case where MOD (%) is valid

Floating Point Expressions

$$\underbrace{\text{float} = \text{float} \begin{matrix} * \\ / \\ + \\ - \end{matrix} \text{float}}_{\text{float}}$$

- TYPE COERCION: When the data type of a value is changed implicitly through mixed-mode arithmetic

Mixed – Mode Expressions

$$\underbrace{\begin{matrix} \text{int} & & \text{float} \\ & \text{or} & \\ \text{float} & & \text{int} \end{matrix} \begin{matrix} * \\ / \\ + \\ - \end{matrix}}_{\text{float}}$$

or

int * float

float / + - int

or

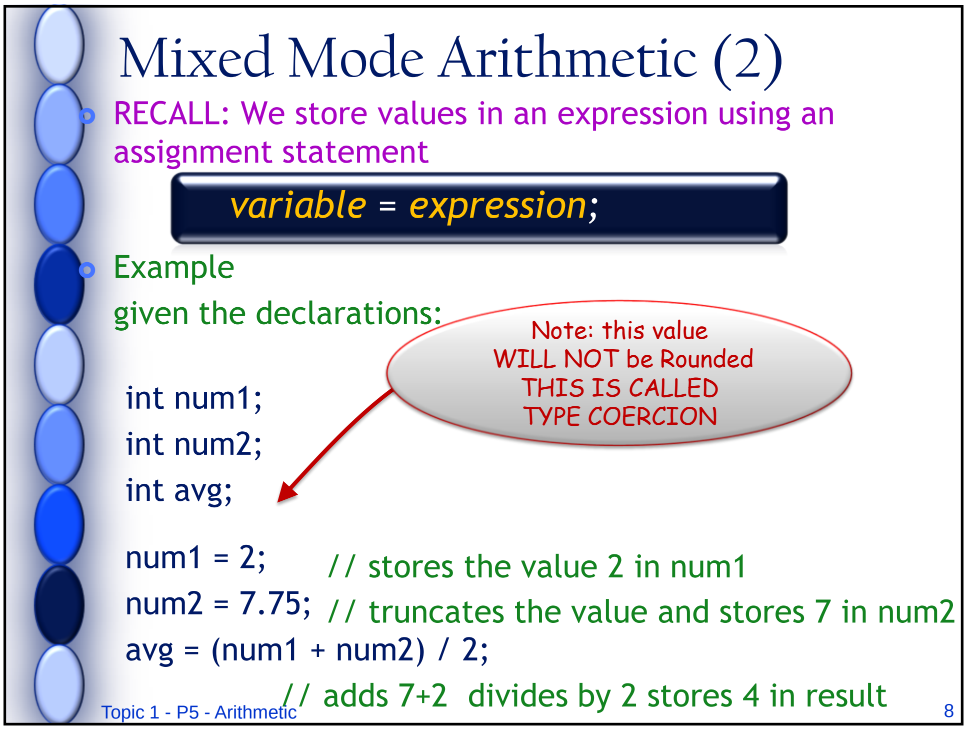
float = int

int = float

Coerces to a float (adds a .0)

Coerces to an int (truncates decimal)

7



Mixed Mode Arithmetic (2)

- RECALL: We store values in an expression using an assignment statement

variable = *expression*;

- Example

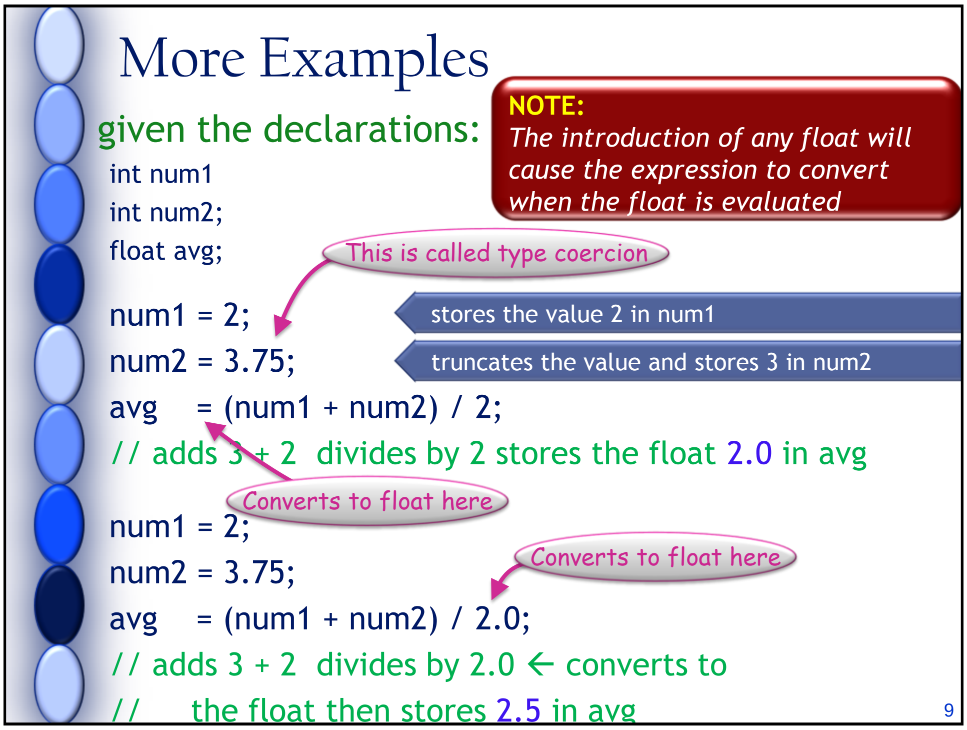
given the declarations:

```
int num1;
int num2;
int avg;

num1 = 2;      // stores the value 2 in num1
num2 = 7.75;   // truncates the value and stores 7 in num2
avg = (num1 + num2) / 2;
               // adds 7+2  divides by 2 stores 4 in result
```

Note: this value WILL NOT be Rounded THIS IS CALLED TYPE COERCION

Topic 1 - P5 - Arithmetic

8



More Examples

given the declarations:

```
int num1
int num2;
float avg;
```

NOTE:
The introduction of any float will cause the expression to convert when the float is evaluated

This is called type coercion

```
num1 = 2;
num2 = 3.75;
avg    = (num1 + num2) / 2;
// adds 3 + 2  divides by 2 stores the float 2.0 in avg
```

stores the value 2 in num1

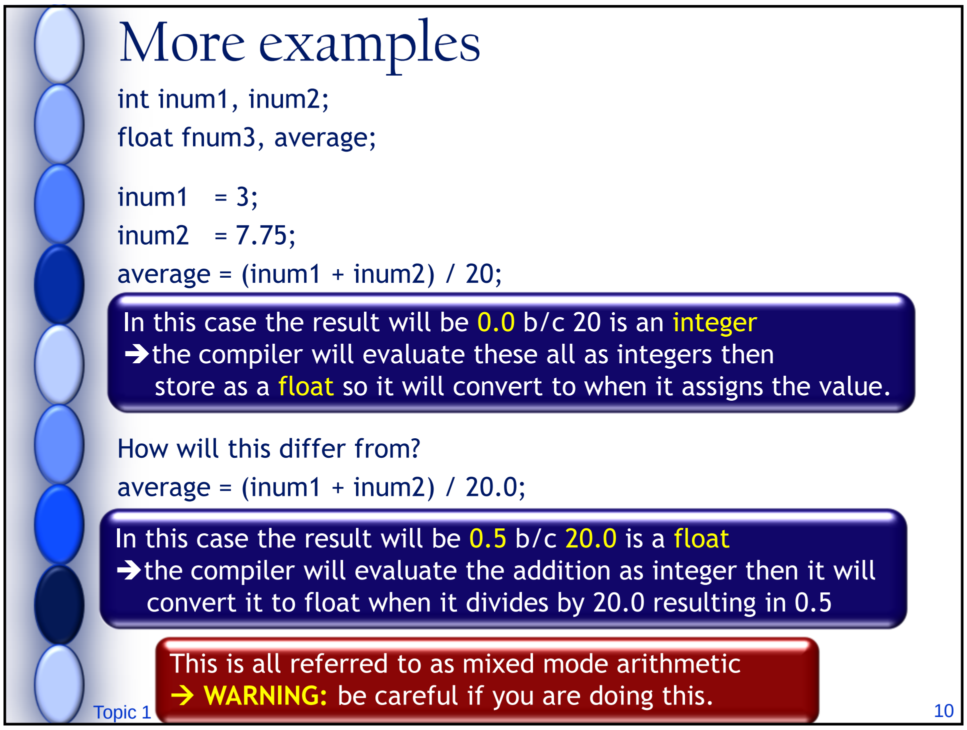
truncates the value and stores 3 in num2

Converts to float here

```
num1 = 2;
num2 = 3.75;
avg    = (num1 + num2) / 2.0;
// adds 3 + 2  divides by 2.0 ← converts to
//     the float then stores 2.5 in avg
```

Converts to float here

9



More examples

```
int inum1, inum2;
float fnum3, average;

inum1   = 3;
inum2   = 7.75;
average = (inum1 + inum2) / 20;
```

In this case the result will be 0.0 b/c 20 is an integer
→the compiler will evaluate these all as integers then store as a float so it will convert to when it assigns the value.

How will this differ from?

```
average = (inum1 + inum2) / 20.0;
```

In this case the result will be 0.5 b/c 20.0 is a float
→the compiler will evaluate the addition as integer then it will convert it to float when it divides by 20.0 resulting in 0.5

This is all referred to as mixed mode arithmetic
→ **WARNING:** be careful if you are doing this.

Topic 1

10

More Examples

Given the declarations:

int inum1, inum2;
float fnum3, average;

What will the compiler do with these assignment statements?

inum1 inum2 fnum3 average

inum1 = 3; 3 is a valid integer ➔ the value 3 is stored in location inum1

inum2 = 7.75; inum2 is an integer so the value 7.75 is truncated (fractional part is cut off) ➔ 7 is stored into inum2
Note: the value is not rounded

fnum3 = 5; 5 is an integer value → when you insert an integer value into a float it converts it to a float.
➔5.0 is stored in fnum3

average = (inum1 + inum2) / 2.0; This will add the 2 integers first b/c of the (). Then it takes the total 10 and divides it by 2.0 converting it to floating point
➔ the floating point value of 5.0. is entered into average

Type Casting

Assume:

int age1, age2, totAge;
float avgAge;
age1 = 2;
age2 = 9;
totAge = 2;

avgAge = float(age1 + age2) / totAge;

- If would add the values age1 and age2, convert them to the floating point value 11.0
- then perform the division producing the desired result 5.5.

Which of these would produce an accurate result?

avgAge = float(age1 + age2) / totAge;
avgAge = (age1 + age2) / float(totAge);
avgAge = (age1 + age2) / 2.0;
avgAge = float((age1 + age2) / totAge);

Assignment Expression

- assignment expression
  - Storing an expression which has a value into a variable

```
variable = expression;
```

- When you add a semi-colon this becomes an expression statement

Multiple Assignments

- Multiple assignments can be used to set several variables to the same value

Example

```
num1 = num2 = num3 = num4 = 0;
```

Embedding Assignment Expressions

- Assignments can also be embedded

Example

```
cout << (num2=10);
```

This performs 2 tasks
→1. it assigns the value 10 into the variable num2
→2. it displays the contents of the variable num2 on the screen

- Assignments are expressions NOT statements
  - They can be used anywhere an expression can be used

Example

```
num2 = 3;
num3 = num2 + 5 * (num1 = 7);
```

This statement is evaluated as follows:

1. num1 is assigned the value 7
   num3 = num2 + 5 * 7
2. The multiplication is evaluated
   num3 = num2 + 35
3. The addition is evaluated
   num3 = 38

Two assignment statements were made in the 2nd statement.
→The value 7 was stored in num1
→The value 38 was stored into num3

WARNING
→Doing this in practice can cause you needless hours debugging!
→And your friends who help you debug will not appreciate it!
→This makes your code confusing to understand
→Bad style

EXAMPLE

```
int in1
int in2;                                   in1      in2      fn3
float fn3;

in1 = ( fn3 = (in2 = 5) * 4 / 8.0 ) * 2;
cout << in1 << endl << in2 << endl << fn3 << endl;
   in1   = (fn3 = 5 * 4 / 8.0) * 2
         = (fn3 = 20 / 8.0) * 2
         = (fn3 = 2.5) * 2
         = 2.5 * 2 = 5.0
   in1   = 5

in1 = ( fn3 = (in2 = 5) * 4 / 8 ) * 2;
cout << in1 << endl << in2 << endl << fn3 << endl;
```

Evaluate this one on your own

EXAMPLE #2

```
                                   in1      in2      fn3      fn4
int in1,in2;
float fn3,fn4;

in1 = ( fn3 = (in2 = 5) * (4 / 8.0) ) * 2;
if (fn4 = (in1 = (in2 * 2) + fn3)) > 10)
{
   cout << fn4;
}
else
{

   cout << "Test val is 10 or less";
}
```

EXERCISE

```
int in1,in2;
float fn3,fn4;
```

in1	in2	fn3	fn4
10	5	0.0	10.0

```
in1 = ( fn3 = (in2 = 5) * (4 / 8) ) * 2;
if ((fn4 = (in1 = (in2 * 2) + fn3)) > 10)
{
   cout << fn4;
}
else
{

   cout << "Test val is 10 or less";
}
```

19

Exercises

Rewrite the following statement

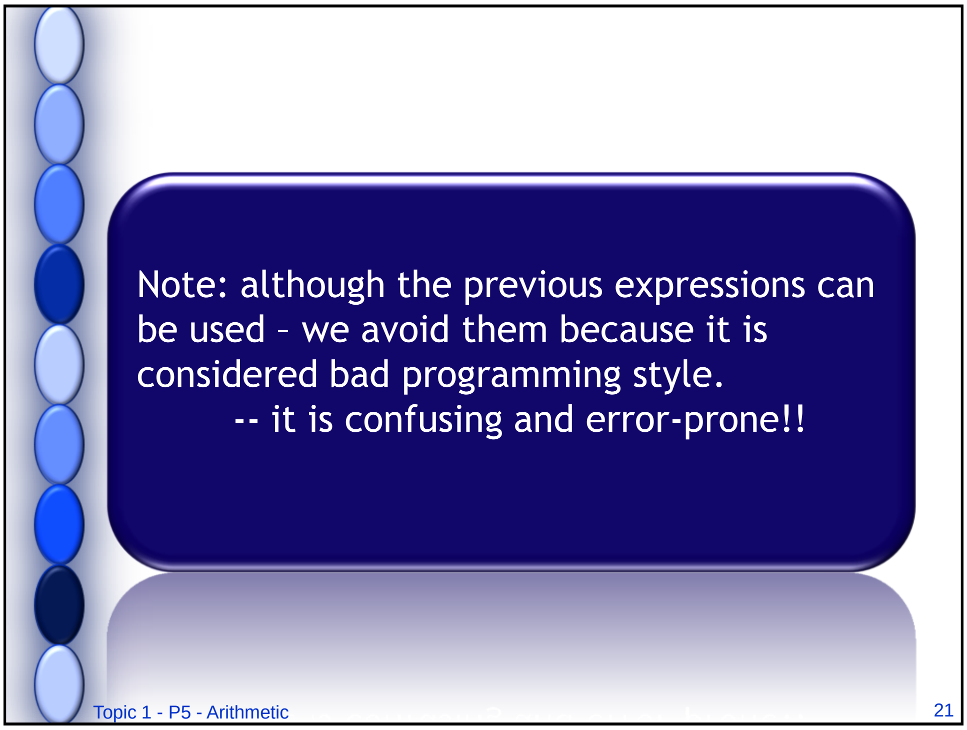
```
if ((x=y) < z)
```

How will this be evaluated?

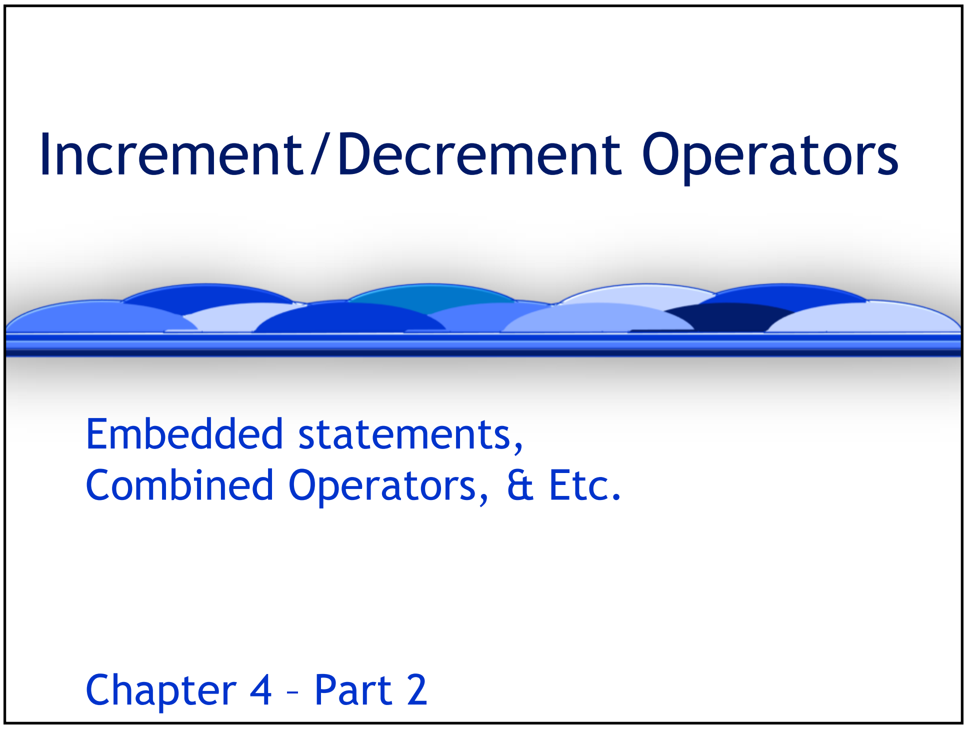
```
If (x = y < z)
```

Remember order of precendence?
Which comes first?

20



Note: although the previous expressions can be used – we avoid them because it is considered bad programming style.
-- it is confusing and error-prone!!



Increment/Decrement Operators

Embedded statements,
Combined Operators, & Etc.

Chapter 4 – Part 2

Increment & Decrement Operators

- Increment & Decrement operators either add 1 or subtract 1
  - Can be used with integer or floating point values
  - Unary operations (1 operand) → single variable
- Syntax

```
++  ➔ Increment
++variable;  or variable++;

--  ➔ Decrement
--variable; or variable--;
```

Example:

age

int age;

age = 20;

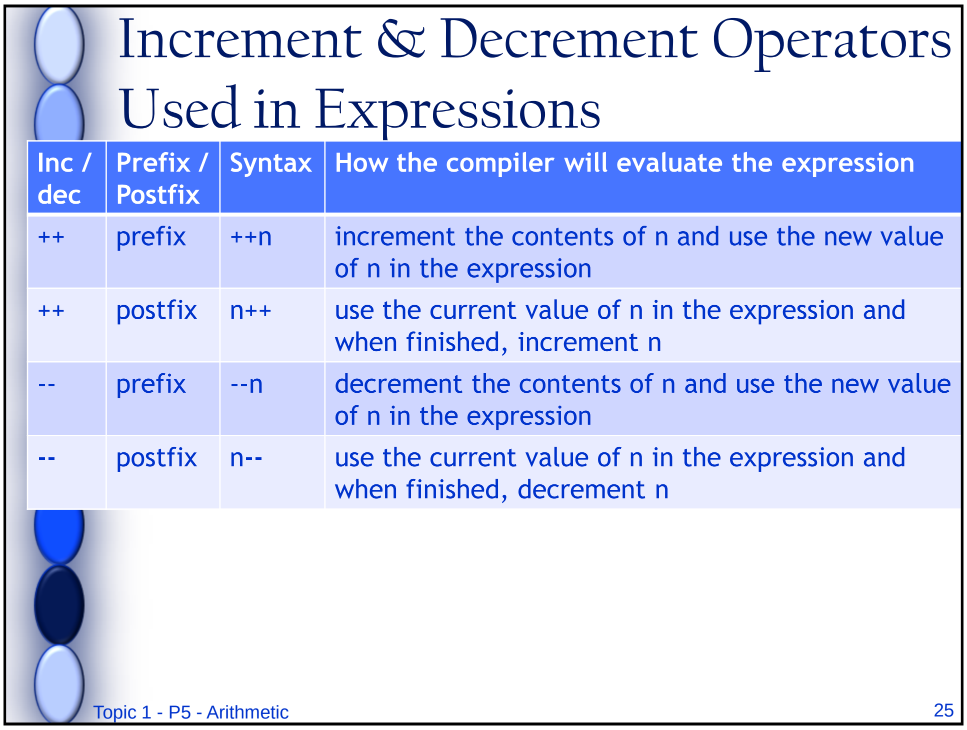
age++; ← This is logically equivalent to age=age+1;

Prefix & Postfix

Prefix → ++age; (or --age;)

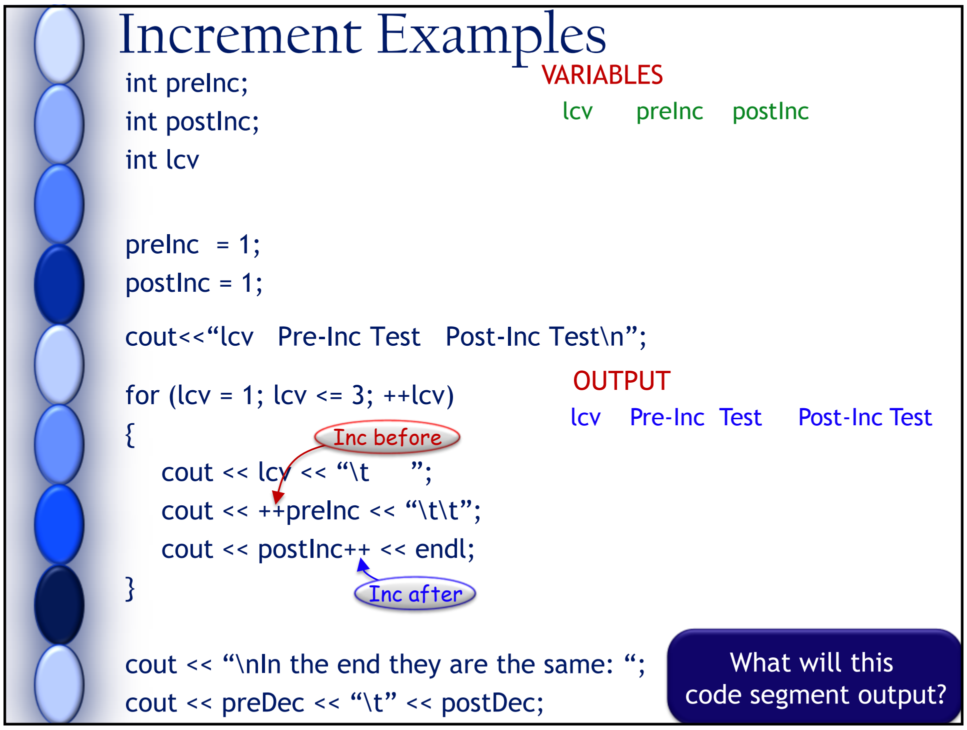
Postfix → age++; (or age--;)

- Using these operators alone will produce the same results
- Using them as part of a larger expression may not
  - → the compiler does not evaluate them the same way



Increment & Decrement Operators Used in Expressions

Inc / dec	Prefix / Postfix	Syntax	How the compiler will evaluate the expression
++	prefix	++n	increment the contents of n and use the new value of n in the expression
++	postfix	n++	use the current value of n in the expression and when finished, increment n
--	prefix	--n	decrement the contents of n and use the new value of n in the expression
--	postfix	n--	use the current value of n in the expression and when finished, decrement n



Increment Examples

```
int preInc;
int postInc;
int lcv

preInc  = 1;
postInc = 1;

cout<<“lcv   Pre-Inc Test   Post-Inc Test\n”;

for (lcv = 1; lcv <= 3; ++lcv)
{
    cout << lcv << “\t     ”;
    cout << ++preInc << “\t\t”;
    cout << postInc++ << endl;
}

cout << “\nIn the end they are the same: “;
cout << preDec << “\t” << postDec;
```

Inc before

Inc after

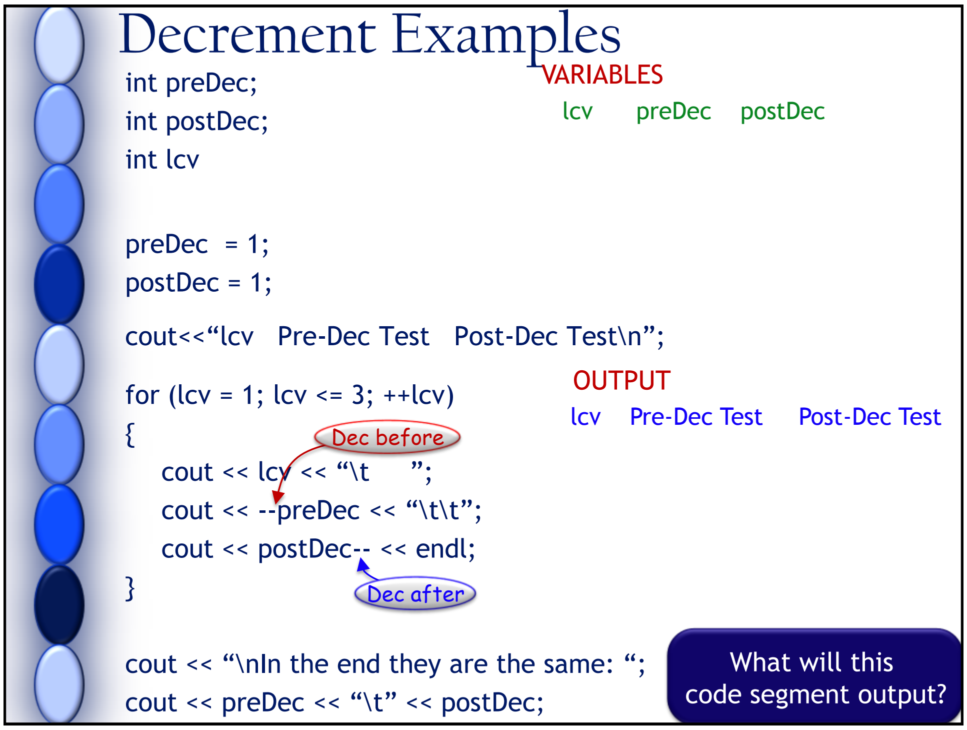
VARIABLES

lcv   preInc   postInc

OUTPUT

lcv   Pre-Inc Test   Post-Inc Test

What will this code segment output?



Decrement Examples

```
int preDec;
int postDec;
int lcv

preDec  = 1;
postDec = 1;

cout<<"lcv   Pre-Dec Test   Post-Dec Test\n";

for (lcv = 1; lcv <= 3; ++lcv)
{
    cout << lcv << "\t     ";
    cout << --preDec << "\t\t";
    cout << postDec-- << endl;
}

cout << "\nIn the end they are the same: ";
cout << preDec << "\t" << postDec;
```

Dec before

Dec after

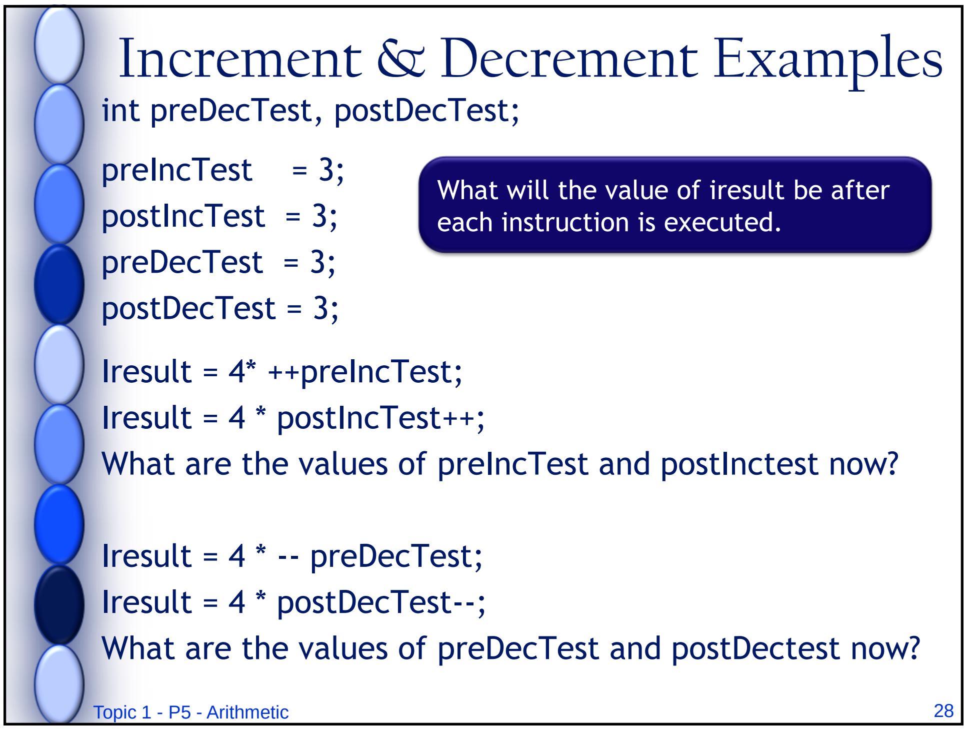
VARIABLES

lcv preDec postDec

OUTPUT

lcv Pre-Dec Test Post-Dec Test

What will this code segment output?



Increment & Decrement Examples

```
int preDecTest, postDecTest;

preIncTest   = 3;
postIncTest  = 3;
preDecTest   = 3;
postDecTest  = 3;

Iresult = 4* ++preIncTest;
Iresult = 4 * postIncTest++;
```

What are the values of preIncTest and postInctest now?

```
Iresult = 4 * -- preDecTest;
Iresult = 4 * postDecTest--;
```

What are the values of preDecTest and postDectest now?

What will the value of iresult be after each instruction is executed.

Combined Operators

- C++ allows operators to be combined
  - Why? → shorthand
  - WARNING: Many environments discourage this
  - It decreases readability → Makes code confusing
- How to use them:

Combination	Syntax	Equivalent to…
+=	num += 5;	num = num + 5;
-=	num -= 3;	num = num - 3;
*=	num *= 10;	num = num * 10;
%=	num %= 2;	num = num % 2;
/=	num /= 2;	num = num / 2;

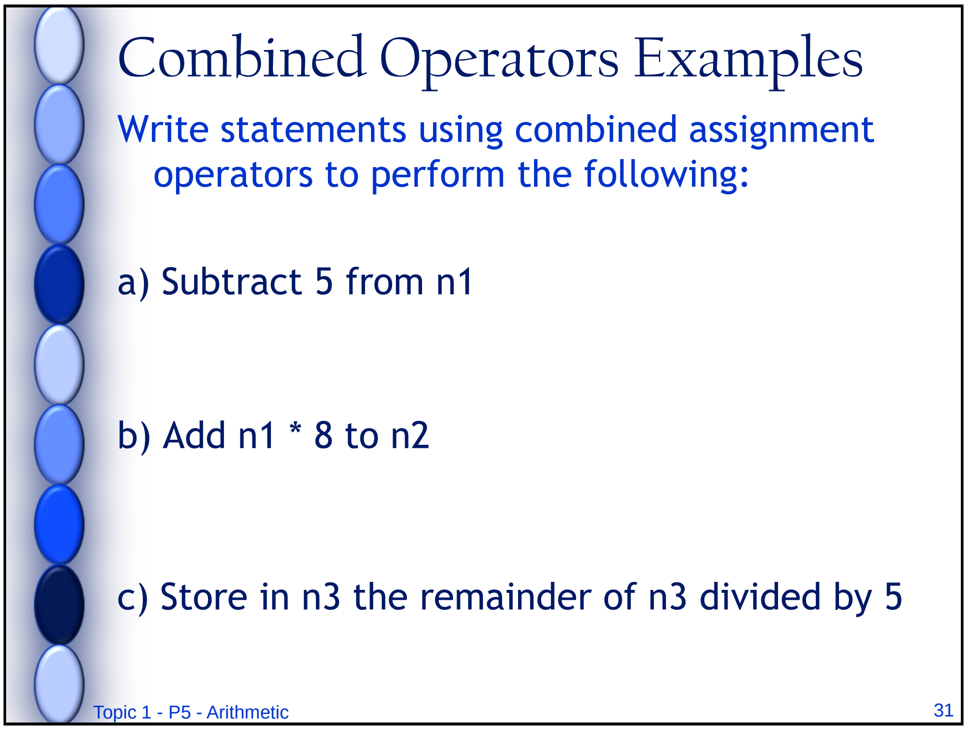
Example: How would we rewrite this?

```
num3 *= num + 10;
```

Note: The precedence of combined ops
Is **lower** than that of the regular math ops.

Order of Precedence (expanded)

Order of Precedence
()
++, --, ! (unary)
*   /   %
+   -
<   <=   >   >=
==   !=
&&
||
=   +=   -=   *=   /=   %=



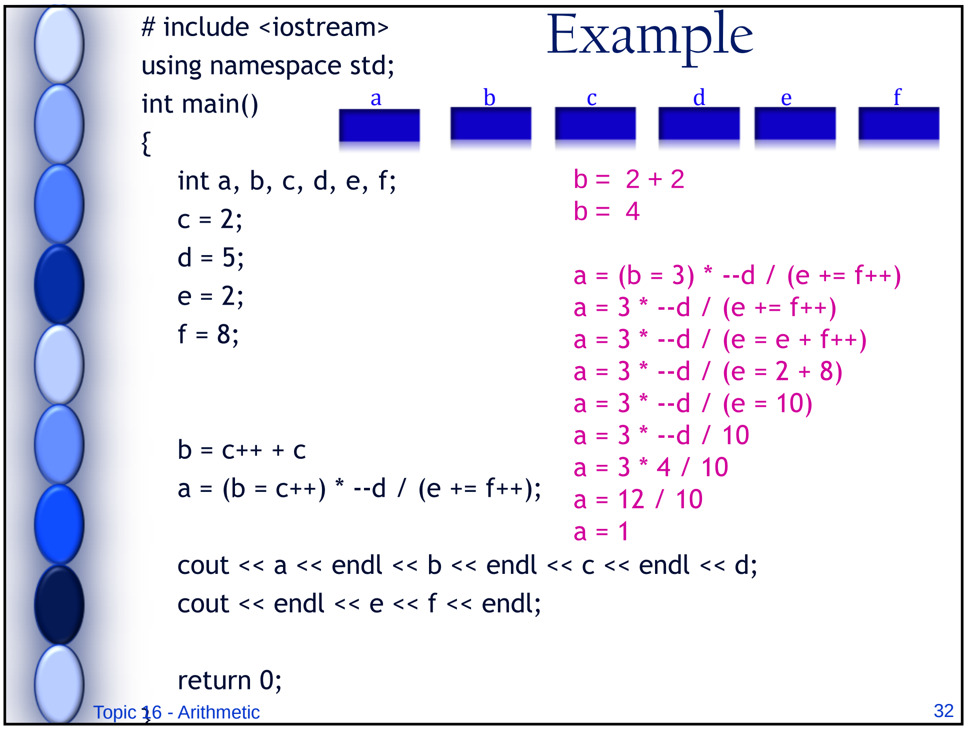
Combined Operators Examples

Write statements using combined assignment operators to perform the following:

a) Subtract 5 from n1

b) Add n1 * 8 to n2

c) Store in n3 the remainder of n3 divided by 5



Example

```
# include <iostream>
using namespace std;
int main()
{
    int a, b, c, d, e, f;
    c = 2;
    d = 5;
    e = 2;
    f = 8;


    b = c++ + c
    a = (b = c++) * --d / (e += f++);
    cout << a << endl << b << endl << c << endl << d;
    cout << endl << e << f << endl;

    return 0;
}
```

```
b =  2 + 2
b =  4

a = (b = 3) * --d / (e += f++)
a = 3 * --d / (e += f++)
a = 3 * --d / (e = e + f++)
a = 3 * --d / (e = 2 + 8)
a = 3 * --d / (e = 10)
a = 3 * --d / 10
a = 3 * 4 / 10
a = 12 / 10
a = 1
```

Exercise #1

a b c d e f

```
# include <iostream>
using namespace std;
int main()
{
    int a, b, c, d, e, f;
    a = 4;
    b = 6;
    c = 3;

    e = (d = c * b++) + --a;
    f = (b += c++);



    cout << a << endl << b << endl << c << endl << d << endl;
    cout << e << endl << f << endl;
    return 0;
}
```

Exercise #2

a b c d e f

```
int main()
{
    int a, b, c, d, e, f;
    a = 2;
    b = 5;
    c =10;

    b *= c;
    d = --b * c++;
    e = --c * (a += 5);
    f = --a + --b * c++;

    cout << a << endl << b;
    cout << endl << c
    cout <<<< endl << d
    cout << endl <<e << endl << f << endl;
    return 0;
}
```