Advanced Arithmetic

CS1A

- Multiple Assignments
- Embedded statements
- Increment / Decrement
- Combined operators

Multiple Assignments

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

Example

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


Embedded 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

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

Evaluate the following assignment expressions.

```
Int   in1
Int   in2;
float fn3;
```

in1 in2 fn3

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

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

Evaluate this one on your own

Evaluate the following assignment expressions.

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

in1 in2 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";
}
```

Evaluate the following assignment expressions.

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

in1 = ( fn3 = (in2 = 5) * (4 / 8) ) * 2;



if ((fn4 = (in1 = (in2 * 2) + fn3)) > 10)
{
    cout << fn4;
}
else
{
   cout << "Test val is 10 or less";
}
```

in1 in2 fn3 fn4

Exercises

Rewrite the following statement → one operation at a time

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

How will this be evaluated?

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

Remember order of precendence? Which comes first?

Note: although the previous expressions can be used - we avoid combining too may expressions because it is considered bad programming style.

-- it is confusing and error-prone!!

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:

```
int age;

age = 20;
age++;
```

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

Using Increment & Decrement 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 << preInc << '\t' << postInc;
```

VARIABLES: lcv preInc postInc

OUTPUT

Inc before

Inc after

Let's do a desk check

Increment & Decrement Examples

```
int preDecTest, postIncTest, preDecTest,postDecTest, ;

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

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

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

What are the values of preIncTest and postInctest now?

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

What are the values of preDecTest and postDectest now?

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 (repeated)

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

Combined Operators Examples

Write statements using combined assignment operators to perform the following:

a) Subtract 5 from n1 & store the result in n1

b) Add n1 * 8 to n2 & store the result in n2

c) Get the remainder of n3 divided by 5 and store the result in n3

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;
   cout << c << endl << d << endl;
   cout << e << endl << f << endl;

   return 0;
}
```

a b c d e f

Exercise #1

```
# 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;
    cout << c << endl << d << endl;
    cout << e << endl << f << endl;

    return 0;
}
```

Exercise #2

```
# include <iostream>
using namespace std;
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 << endl;
    cout << c << endl << d << endl;
    cout << e << endl << f << endl;


    return 0;
}
```