Display Time: In-class

- Write a program that obtains hours, minutes and remaining seconds from seconds.
- Get the time as seconds from the user and convert it to hours, minutes and seconds

3.2

Mathematical Expressions

Mathematical Expressions

- Can create complex expressions using multiple mathematical operators
- An expression can be a literal, a variable, or a mathematical combination of constants and variables
- Can be used in assignment, `cout`, other statements:

```
area = 2 * PI * radius;
cout << "border is: " << 2*(l+w);
```

Algebraic Expressions

- Multiplication requires an operator:

  Area=*lw* is written as `Area = l * w;`

- There is no exponentiation operator:

  $Area = s^2$ is written as `Area = pow(s, 2);`

- Parentheses may be needed to maintain order of operations:

  $$m = \frac{y2 - y1}{x2 - x1}$$

  is written as

  ```
  m = (y2-y1) /(x2-x1);
  ```

Algebraic Expressions

Table 3-5 Algebraic and C++ Multiplication Expressions

Algebraic Expression	Operation	C++ Equivalent
$6B$	6 times B	6 * B
(3)(12)	3 times 12	3 * 12
$4xy$	4 times x times y	4 * x * y

Exponent Operations

- Need to include cmath library for this operation

```
cout << pow(2.0, 3) << endl; // Display 8.0
cout << pow(4.0, 0.5) << endl; // Display 2.0
cout << pow(2.5, 2) << endl; // Display 6.25
cout << pow(2.5, -2) << endl; // Display 0.16
```

- 3.5.cpp

X ** Y

Write a program that reads the values of x and y from the keyboard, and implement the following formula:

$$Value = x \hat{} y$$

Operator Precedence

- Mathematical expressions can be very complex.
- There is a set order in which arithmetic operations will be carried out.

	Operator	Associativity	Example	Result
Higher Priority	– (unary negation)	Right to left	`x = -4 + 3;`	-1
	`* / %`	Left to right	`x = -4 + 4 % 3 * 13 + 2;`	11
Lower Priority	`+ –`	Left to right	`x = 6 + 3 – 4 + 6 * 3;`	23

Precedence rules for arithmetic operators.

Operator/Convention	Description	Explanation
( )	Items within parentheses are evaluated first	In 2 * (x + 1), the x + 1 is evaluated first, with the result then multiplied by 2.
unary -	- used for negation (unary minus) is next	In 2 * -x, the -x is computed first, with the result then multiplied by 2.
*** / %**	Next to be evaluated are *, /, and %, having equal precedence.	In y = 3 + 14%3, 14%3 is evaluated first, with the result then added to 3, because % has higher precedence than +.
+ -	Finally come + and - with equal precedence.	In y = 3 + 2 * x, the 2 * x is evaluated first, with the result then added to 3, because * has higher precedence than +. Spacing doesn't matter: y = 3+2 * x would still evaluate 2 * x first.
left-to-right	If more than one operator of equal precedence could be evaluated, evaluation occurs left to right.	In y = x * 2 / 3, the x * 2 is first evaluated, with the result then divided by 3.

Associativity of Operators

- `-` (unary negation) associates right to left
- `*, /, %, +, -` associate right to left
- parentheses ( ) can be used to override the order of operations:

```
 2 + 2  *  2 - 2  = 4
(2 + 2) *  2 - 2  = 6
 2 + 2  * (2 - 2) = 2
(2 + 2) * (2 - 2) = 0
```

Order of Operations

In an expression with more than one operator, evaluate in this order:

- `-` (unary negation), in order, left to right
- `*` `/` `%`, in order, left to right
- `+` `-`, in order, left to right

In the expression `2 + 2 * 2 - 2`

Arithmetic Expressions

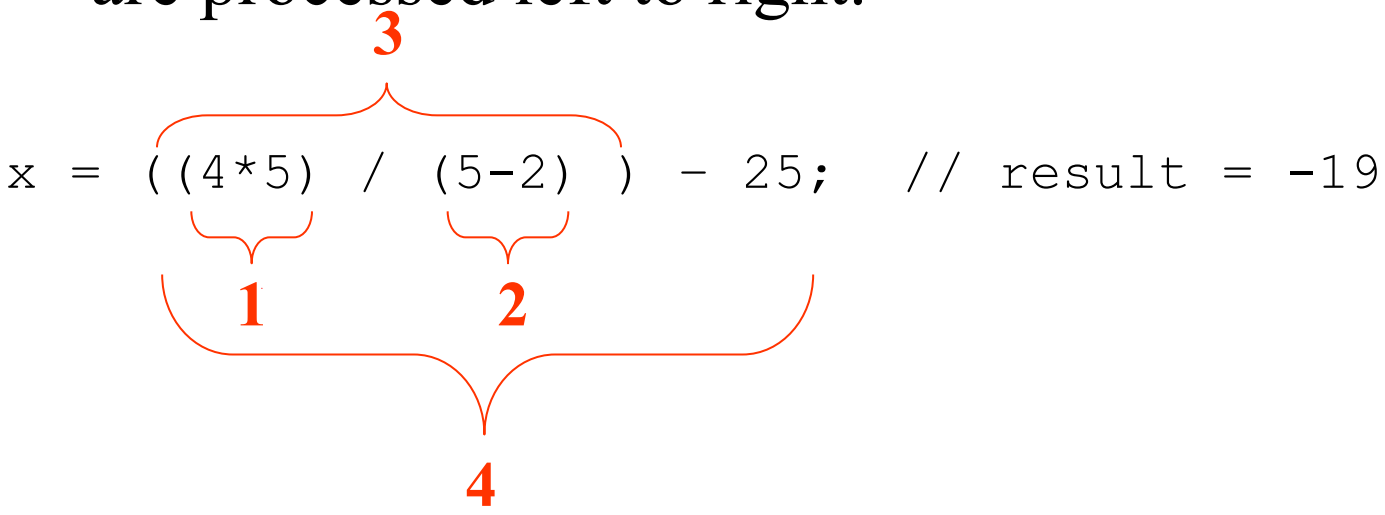
$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

is translated to

(3+4*x)/5 – (10*(y-5)*(a+b+c))/x + 9*(4/x + (9+x)/y)

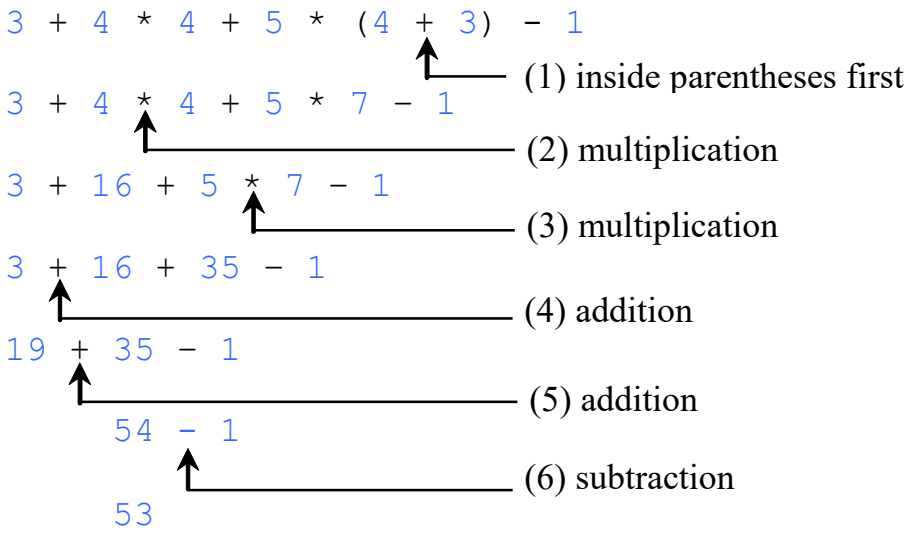
Grouping with Parenthesis

- When parenthesis are used in an expression, the inner most parenthesis are processed first.
- If two sets of parenthesis are at the same level, they are processed left to right.



How to Evaluate an Expression

Though C++ has its own way to evaluate an expression behind the scene, the result of a C++ expression and its corresponding arithmetic expression are the same. Therefore, you can safely apply the arithmetic rule for evaluating a C++ expression.



Grouping with Parentheses

Table 3-4 More Simple Expressions and Their Values

Expression	Value
(5 + 2) * 4	28
10 / (5 − 3)	5
8 + 12 * (6 − 2)	56
(4 + 17) % 2 − 1	0
(6 − 3) * (2 + 7) / 3	9

Grouping with Parentheses

Example:

How would we encode the following formula in C?

$$y = \frac{4ac - d}{2 - b}$$

Grouping with Parentheses

Example:

How would we encode the following formula in C?

$$y = \frac{4ac - d}{2 - b}$$

We could do it as follows:

```
y = (4 * a * c - d) / (2 - b);
```

Grouping with Parentheses

Example:

How would we encode the following formula in C?

$$y=\frac{4ac-d}{2-b}$$

We could do it as follows:

```
y = (4 * a * c - d) / (2 - b);
```

If we remove the parentheses from the statement above, the implemented equation is:

$$y = 4ac - \frac{d}{2} - b$$

Simple Equations

A quadratic (second order polynomial) equation

$$ax^2 + bx + c = 0$$

can be solved by the formulas

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Note: The square root of a negative value is a complex number. In this case, VC++ throws an error like: `-1.#IND`

Write a program to solve for x in equation $x^2 - 5x + 6 = 0$.

Simple Equations

Program1: Solve for $x^2 - 5x + 6 = 0$

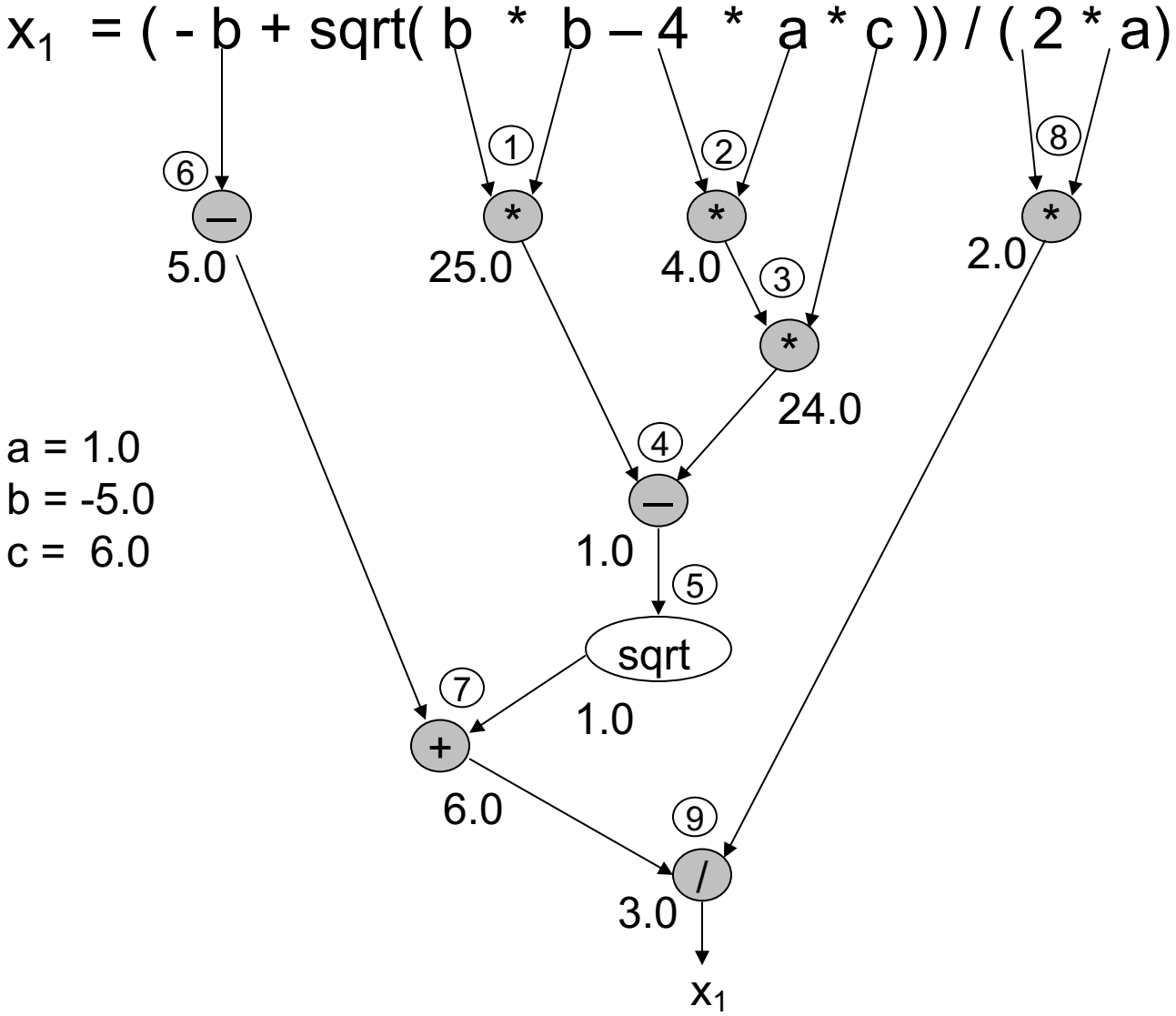
```
 /* File: secondorder1.c */
#include <iostream>
#include <math.h>

 int main() {
     double a = 1.0, b = -5.0, c = 6.0, x1, x2;
     x1 = (-b +sqrt(b*b-4*a*c))/(2*a);
     x2 = (-b -sqrt(b*b-4*a*c))/(2*a);
    cout << "x1 = " << x1 << endl;
    cout << "x2 = " << x2 << endl;
     return 0;
 }
```

Output:

```
x1 = 3.000000
x2 = 2.000000
```

Simple Equations

quadratic.cpp



3.3

When You Mix Apples with Oranges: Type Conversion

When You Mix Apples with Oranges: Type Conversion

- Operations are performed between operands of the same type.
- If not of the same type, C++ will convert one to be the type of the other
- This can impact the results of calculations.

Hierarchy of Types

Highest:

```
long double
double
float
unsigned long
long
unsigned int
int
```

Lowest:

Ranked by largest number they can hold

Type Coercion

- Type Coercion: automatic conversion of an operand to another data type
- Promotion: convert to a higher type
- Demotion: convert to a lower type

Coercion Rules

1) char, short, unsigned short automatically promoted to int

2) When operating on values of different data types, the lower one is promoted to the type of the higher one.

3) When using the = operator, the type of expression on right will be converted to type of variable on left

Coercion.cpp

Coercion Rules

1) char, short, unsigned short automatically promoted to int

2) When operating on values of different data types, the lower one is promoted to the type of the higher one.

3) When using the = operator, the type of expression on right will be converted to type of variable on left

Coercion.cpp

int x = y * 2.4

3 * 2.4

x = 7

3.0 * 2.4 = 7.2

3.4

Overflow and Underflow

Overflow and Underflow

- Occurs when assigning a value that is too large (overflow) or too small (underflow) to be held in a variable
- Variable contains value that is 'wrapped around' set of possible values
- Different systems may display a warning/error message, stop the program, or continue execution using the incorrect value

- 3.7.cpp, overUnderFlow.cpp

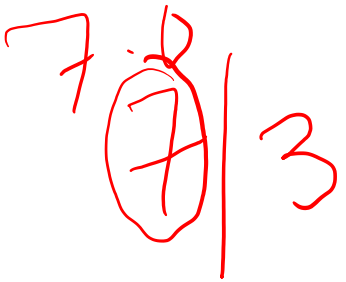
3.5

Type Casting

Type Casting

- Used for manual data type conversion
- Useful for floating point division using ints:

```
double m;
m = static_cast<double>(y2-y1)
                        /(x2-x1);
```



- Useful to see `int` value of a `char` variable:

```
char ch = 'C';
cout << ch << " is "
     << static_cast<int>(ch);
```

Type Casting in Program 3-9

Program 3-9

```
1  // This program uses a type cast to avoid integer division.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     int books;         // Number of books to read
8     int months;        // Number of months spent reading
9     double perMonth;   // Average number of books per month
10
11    cout << "How many books do you plan to read? ";
12    cin >> books;
13    cout << "How many months will it take you to read them? ";
14    cin >> months;
15    perMonth = static_cast<double>(books) / months;
16    cout << "That is " << perMonth << " books per month.\n";
17    return 0;
18 }
```

Program Output with Example Input Shown in Bold

```
How many books do you plan to read? 30 [Enter]
How many months will it take you to read them? 7 [Enter]
That is 4.28571 books per month.
```

C-Style and Prestandard Type Cast Expressions

- C-Style cast: data type name in `()`

```
cout << ch << " is " << (int)ch;
```

- Prestandard C++ cast: value in `()`

```
cout << ch << " is " << int(ch);
```

- Both are still supported in C++, although `static_cast` is preferred

- prodAddDiffStatic.cpp

Numeric Type Conversion

Consider the following statements:

```
short i = 100;
long k = i * 3 + 4;
double d = i * 3.1 + k / 2;
```

Type Casting

- Implicit casting

  double d = 3; (type widening)

- Explicit casting

  int i = static_cast<int>(3.0); (type narrowing)

- **int i = (int)3.9;** (Fraction part is truncated)

Promotion (Implicit Conversion)

- The algorithms and resultant data types of operations depend on the data types of the operands.
- A data type that occupies less memory can be converted to a data type that occupies more memory space without loss of any information.
- For binary operations, such as addition, subtraction, multiplication, and division, the resultant data type will take the higher order data type of two operands.
  - For example, the addition of two float numbers will result in a float, while the addition of a float and a double will result in a double.
- This is called ***implicit conversion*** (i.e., performed by C++ automatically).

Promotion (Implicit Conversion)

- The type of each value in a mixed-type expression is automatically promoted to the "highest" type in the expression (actually a temporary version of each value is created and used for the expression—the original values remain unchanged). .
- Converting values to lower types (demotion) can result in an incorrect value under certain conditions. Many compilers, therefore, will throw a warning if this is attempted.
- In spite of that, C++ will generally perform demotions (e.g., in assignments and function calls) without having to use a cast, although a cast (explicit conversion) can always be used.

Promotion (Implicit Conversion)

Data type	printf conversion specification	scanf conversion specification
Floating-point types		
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f

Fig. 5.5 | Arithmetic data types and their conversion specifications. (Part 1 of 2.)

Data type	printf conversion specification	scanf conversion specification
Integer types		
unsigned long long int	%llu	%llu
long long int	%lld	%lld
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
unsigned short	%hu	%hu
short	%hd	%hd
char	%c	%c

Fig. 5.5 | Arithmetic data types and their conversion specifications. (Part 2 of 2.)

NOTE

Casting does not change the variable being cast. For example, d is not changed after casting in the following code:

```
double d = 4.5;
int i = static_cast<int>(d);  // d is not changed
```

staticCast.cpp

Casting (Explicit Conversion)

- Cast operators are available for most data types.
- The cast operator is formed by placing parentheses around a data type name.
  - `average = ( double ) total / counter;`
- The cast operator is a unary operator, i.e., an operator that takes only one operand.
- Cast operators associate from right to left and have the same precedence as other unary operators such as unary + and unary -.
  - This precedence is one level higher than that of the multiplicative operators *, / and %.

Casting (Explicit Conversion)

- Casting can be used to force non-integer division when integer variables are used.
- Example:

```
int x = 22, y = 8;
double g;
g = (double) x / y;
```

  - g will contain 2.75, since one of the operands, x, was converted to a double. Therefore, the division was not integer division.
- Be careful, however. If the cast operator is applied to the entire expression, the result will be different:

```
g = (double) (x / y);
```

  - Now g contains 2.0. The division was performed as an integer division (result = 2), that result was converted to a double (result = 2.0), and then assigned to "g".

Compute Average

- Reading multiple inputs in one statement
- Calculate average of three numbers by taking in the input from the users.

Example: Converting Temperatures

Write a program that converts a Fahrenheit degree to Celsius using the formula:

$$celsius = \frac{5}{9} * fahrenheit - 32$$

Example: Converting Temperatures

Write a program that converts a Fahrenheit degree to Celsius using the formula:

$$celsius = (\frac{5}{9})(fahrenheit - 32)$$

Example: Keeping Two Digits After Decimal Points

The sales tax is 6% for any product purchased. Write a program that displays the sales tax with two digits after the decimal point.

SalesTaxDecimalPoint.cpp

Problem: Monetary Units

Calculate the number of $5 notes and $1 notes that a person can have if the user enters a value. Your program should report maximum number of five dollars followed by one dollar

computingChange.cpp

Problem: Monetary Units

This program lets the user enter the amount in decimal representing dollars and cents and output a report listing the monetary equivalent in single dollars, quarters, dimes, nickels, and pennies. Your program should report maximum number of dollars, then the maximum number of quarters, and so on, in this order.

computeChange.cpp

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount 1156

remainingAmount initialized

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount	1156
numberOfOneDollars	11

numberOfOneDollars assigned

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount	56
numberOfOneDollars	11

remainingAmount updated

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount	56
numberOfOneDollars	11
numberOfOneQuarters	2

numberOfOneQuarters assigned

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount	6
numberOfOneDollars	11
numberOfQuarters	2

remainingAmount updated