

04 - Arithmetic

Dr. Robert Lowe

Division of Mathematics and Computer Science
Maryville College

Outline

- 1 Types
- 2 Arithmetic Operators
- 3 Programming With Operators

Outline

- 1 Types
- 2 Arithmetic Operators
- 3 Programming With Operators

Variable Types

- C++ has the following variable types:

`bool` Stores a value that is either true or false

`char` Stores a single character (a letter, digit, or any other symbol)

`int` Stores an integer

`float` Stores a single precision floating point number (don't use these!)

`double` Stores a double precision floating point number.

- Variables must be declared before they are used:

```
int x;
```

```
char letter;
```

```
double num;
```

Literals

- A literal is a value that is typed into a program.

Literals

- A literal is a value that is typed into a program.
- Like variables, literals have types.

Literals

- A literal is a value that is typed into a program.
- Like variables, literals have types.
- The compiler infers literal types from the format of the literal.

Literals

- A literal is a value that is typed into a program.
- Like variables, literals have types.
- The compiler infers literal types from the format of the literal.

Example Literals

Literals

- A literal is a value that is typed into a program.
- Like variables, literals have types.
- The compiler infers literal types from the format of the literal.

Example Literals

```
bool true or false
```

Literals

- A literal is a value that is typed into a program.
- Like variables, literals have types.
- The compiler infers literal types from the format of the literal.

Example Literals

```
bool true or false
```

```
char 'a', 'b', '+'
```

Literals

- A literal is a value that is typed into a program.
- Like variables, literals have types.
- The compiler infers literal types from the format of the literal.

Example Literals

```
bool true or false
```

```
char 'a', 'b', '+'
```

```
int 5, 10, 15, 42
```

Literals

- A literal is a value that is typed into a program.
- Like variables, literals have types.
- The compiler infers literal types from the format of the literal.

Example Literals

```
bool true or false
```

```
char 'a', 'b', '+'
```

```
int 5, 10, 15, 42
```

```
float 1.5f, 1.0f
```

Literals

- A literal is a value that is typed into a program.
- Like variables, literals have types.
- The compiler infers literal types from the format of the literal.

Example Literals

```
bool true or false
```

```
char 'a', 'b', '+'
```

```
int 5, 10, 15, 42
```

```
float 1.5f, 1.0f
```

```
double 1.5, 1.0
```

Literals

- A literal is a value that is typed into a program.
- Like variables, literals have types.
- The compiler infers literal types from the format of the literal.

Example Literals

```
bool true or false
char 'a', 'b', '+'
int 5, 10, 15, 42
float 1.5f, 1.0f
double 1.5, 1.0
string "This is a string literal"
```

More About Variable Declarations

- You can declare multiple variables of the same type in one statement.

```
int x, y;
```

```
double a, b;
```

More About Variable Declarations

- You can declare multiple variables of the same type in one statement.

```
int x, y;  
double a, b;
```

- Variables can be assigned initial values (initialized) during declaration.

```
int count=0;  
char fi='R', mi='E', li='L';
```


Constants

- A constant is like a named literal.

Constants

- A constant is like a named literal.
- Unlike a variable, a constant cannot be changed. (So it's not just a clever name!)

Constants

- A constant is like a named literal.
- Unlike a variable, a constant cannot be changed. (So it's not just a clever name!)
- A constant is declared just like a variable, but with the `const` keyword.

Constants

- A constant is like a named literal.
- Unlike a variable, a constant cannot be changed. (So it's not just a clever name!)
- A constant is declared just like a variable, but with the `const` keyword.
- The value of a constant must be immediately assigned when it is declared.

Constants

- A constant is like a named literal.
- Unlike a variable, a constant cannot be changed. (So it's not just a clever name!)
- A constant is declared just like a variable, but with the `const` keyword.
- The value of a constant must be immediately assigned when it is declared.

Constant Example

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

Code Style Notes

- Variable names should begin with a lower case letter.

Code Style Notes

- Variable names should begin with a lower case letter.
- When a variable name has more than one word, either camelCase it or use_underscores.

Code Style Notes

- Variable names should begin with a lower case letter.
- When a variable name has more than one word, either `camelCase` it or `use_underscores`.
- Never mix and match camel casing and underscore variable naming in the same program!

Code Style Notes

- Variable names should begin with a lower case letter.
- When a variable name has more than one word, either `camelCase` it or `use_underscores`.
- Never mix and match camel casing and underscore variable naming in the same program!
- Constants should be named in all upper case letters.

Code Style Notes

- Variable names should begin with a lower case letter.
- When a variable name has more than one word, either `camelCase` it or `use_underscores`.
- Never mix and match camel casing and underscore variable naming in the same program!
- Constants should be named in all upper case letters.
- Always use underscores with constant names with more than one word.

Code Style Notes

- Variable names should begin with a lower case letter.
- When a variable name has more than one word, either `camelCase` it or `use_underscores`.
- Never mix and match camel casing and underscore variable naming in the same program!
- Constants should be named in all upper case letters.
- Always use underscores with constant names with more than one word.
- Use descriptive variable names, but try to keep it short.

Code Style Notes

- Variable names should begin with a lower case letter.
- When a variable name has more than one word, either camelCase it or use_underscores.
- Never mix and match camel casing and underscore variable naming in the same program!
- Constants should be named in all upper case letters.
- Always use underscores with constant names with more than one word.
- Use descriptive variable names, but try to keep it short.
- Only use multi-variable declarations where variables are related. For example, this is fine:

```
double x, y; //coordinates
```

But this is probably not fine:

```
int count, length; //count and length
```

Outline

- 1 Types
- 2 Arithmetic Operators
- 3 Programming With Operators

Integer Arithmetic

+ Addition

Integer Arithmetic

- + Addition
- − Subtraction

Integer Arithmetic

- + Addition
- − Subtraction
- * Multiplication

Integer Arithmetic

- + Addition
- − Subtraction
- * Multiplication
- / Division

Integer Arithmetic

- + Addition
- − Subtraction
- * Multiplication
- / Division
- % Modulus (remainder)

Integer Arithmetic

- + Addition
- − Subtraction
- * Multiplication
- / Division
- % Modulus (remainder)

- Note that when doing integer arithmetic, C++ truncates any fractional parts.

Integer Arithmetic

- + Addition
- − Subtraction
- * Multiplication
- / Division
- % Modulus (remainder)

- Note that when doing integer arithmetic, C++ truncates any fractional parts.
- Division works like “grade school long division”

Integer Arithmetic

- + Addition
- − Subtraction
- * Multiplication
- / Division
- % Modulus (remainder)

- Note that when doing integer arithmetic, C++ truncates any fractional parts.
- Division works like “grade school long division”
- The operator / simply returns the quotient.

Integer Arithmetic

- + Addition
- Subtraction
- * Multiplication
- / Division
- % Modulus (remainder)

- Note that when doing integer arithmetic, C++ truncates any fractional parts.
- Division works like “grade school long division”
- The operator / simply returns the quotient.
- The modulus operator % returns the remainder of the division.

Integer Arithmetic

- + Addition
- − Subtraction
- * Multiplication
- / Division
- % Modulus (remainder)

- Note that when doing integer arithmetic, C++ truncates any fractional parts.
- Division works like “grade school long division”
- The operator / simply returns the quotient.
- The modulus operator % returns the remainder of the division.
- Integer arithmetic is performed on any expression consisting of integer literals or variables.

Floating Point Arithmetic

+ Addition

Floating Point Arithmetic

- + Addition
- − Subtraction

Floating Point Arithmetic

- + Addition
- − Subtraction
- * Multipliation

Floating Point Arithmetic

- + Addition
- − Subtraction
- * Multipliation
- / Division

Floating Point Arithmetic

- + Addition
- − Subtraction
- * Multipliation
- / Division

- Floating point arithmetic is what a pocket calculator typically does.

Floating Point Arithmetic

- + Addition
- − Subtraction
- * Multipliation
- / Division

- Floating point arithmetic is what a pocket calculator typically does.
- This deals with real numbers, so they have fractional parts.

Floating Point Arithmetic

- + Addition
- − Subtraction
- * Multipliation
- / Division

- Floating point arithmetic is what a pocket calculator typically does.
- This deals with real numbers, so they have fractional parts.
- There is no modulus for floating point arithmetic.

Floating Point Arithmetic

- + Addition
- − Subtraction
- * Multipliation
- / Division

- Floating point arithmetic is what a pocket calculator typically does.
- This deals with real numbers, so they have fractional parts.
- There is no modulus for floating point arithmetic.
- Floating point arithmetic is performed on any expression which contains at least one `double` or `float` literal/variable.

Assignment Operators

= Assignment

Assignment Operators

- = Assignment
- += Addition Assignment

Assignment Operators

- = Assignment
- += Addition Assignment
- = Subtraction Assignment

Assignment Operators

- = Assignment
- += Addition Assignment
- = Subtraction Assignment
- *= Multiply Assignment

Assignment Operators

- = Assignment
- += Addition Assignment
- = Subtraction Assignment
- *= Multiply Assignment
- /= Divide Assignment

Assignment Operators

- = Assignment
- += Addition Assignment
- = Subtraction Assignment
- *= Multiply Assignment
- /= Divide Assignment
- %= Modulus Assignment

Assignment Operators

- = Assignment
 - += Addition Assignment
 - = Subtraction Assignment
 - *= Multiply Assignment
 - /= Divide Assignment
 - %= Modulus Assignment
- The left hand side of an assignment operator must be a variable.

Assignment Operators

- = Assignment
 - += Addition Assignment
 - = Subtraction Assignment
 - *= Multiply Assignment
 - /= Divide Assignment
 - %= Modulus Assignment
- The left hand side of an assignment operator must be a variable.
 - Assignment operators change the value of a variable.

Assignment Operators

- = Assignment
 - += Addition Assignment
 - = Subtraction Assignment
 - *= Multiply Assignment
 - /= Divide Assignment
 - %= Modulus Assignment
- The left hand side of an assignment operator must be a variable.
 - Assignment operators change the value of a variable.
 - Assignment operators return the value that was assigned.

Assignment Operators

- = Assignment
 - += Addition Assignment
 - = Subtraction Assignment
 - *= Multiply Assignment
 - /= Divide Assignment
 - %= Modulus Assignment
- The left hand side of an assignment operator must be a variable.
 - Assignment operators change the value of a variable.
 - Assignment operators return the value that was assigned.
 - Compound assignment operators are short-hand ways to modify variables. For example:
 $x \ += \ 1$ is short for $x \ = \ x \ + \ 1$

Operator Precedence

Operator	Description	Associativity
$a * b$, a / b , $a \% b$	Multiply, Divide, Modulus	Left-to-Right
$a + b$, $a - b$	Addition and Subtraction	Left-to-Right
\ll , \gg	Insertion and Extraction	Left-to-Right
$=$, $+=$, $-=$ $*=$, $/=$ $\%=$	Assignment and Assignment	Right-to-Left

- Precedence specifies the order of operations.

Operator Precedence

Operator	Description	Associativity
$a * b$, a / b , $a \% b$	Multiply, Divide, Modulus	Left-to-Right
$a + b$, $a - b$	Addition and Subtraction	Left-to-Right
\ll , \gg	Insertion and Extraction	Left-to-Right
$=$, $+=$, $-=$ $*=$, $/=$ $\%=$	Assignment and Assignment	Right-to-Left

- Precedence specifies the order of operations.
- Associativity is how we “break ties”.

Operator Precedence

Operator	Description	Associativity
$a * b$, a / b , $a \% b$	Multiply, Divide, Modulus	Left-to-Right
$a + b$, $a - b$	Addition and Subtraction	Left-to-Right
\ll , \gg	Insertion and Extraction	Left-to-Right
$=$, $+=$, $-=$ $*=$, $/=$ $\%=$	Assignment and Assignment	Right-to-Left

- Precedence specifies the order of operations.
- Associativity is how we “break ties”.
- Parenthesis can also be used to control order of operations (as in normal math).

Example: pmdas.cpp

```
#include <iostream>

using namespace std;

int main()
{
    cout << "3+2*6=" << 3+2*6 << endl
         << "(3+2)*6=" << (3+2)*6 << endl
         << "5%2=" << 5%2 << endl
         << "6/2*(1+2)=" << 6/2*(1+2) << endl
         << "1/2*4=" << 1/2*4 << endl
         << "1.0/2.0*4=" << 1.0/2.0*4 << endl;
}
```

Statement Resolution

- C++ Resolves statements via expression substitution.

Statement Resolution

- C++ Resolves statements via expression substitution.
- Once all operations are cleared, the statement is completed.

Statement Resolution

- C++ Resolves statements via expression substitution.
- Once all operations are cleared, the statement is completed.
- For example:

Statement Resolution

- C++ Resolves statements via expression substitution.
- Once all operations are cleared, the statement is completed.
- For example:
 $6 / 2 * (1 + 2)$

Statement Resolution

- C++ Resolves statements via expression substitution.
- Once all operations are cleared, the statement is completed.
- For example:

$6 / 2 * (1 + 2)$

$6 / 2 * 3$

Statement Resolution

- C++ Resolves statements via expression substitution.
- Once all operations are cleared, the statement is completed.
- For example:
 $6 / 2 * (1 + 2)$
 $6 / 2 * 3$
 $3 * 3$

Statement Resolution

- C++ Resolves statements via expression substitution.
- Once all operations are cleared, the statement is completed.
- For example:
 $6 / 2 * (1 + 2)$
 $6 / 2 * 3$
 $3 * 3$
 9

Statement Resolution

- C++ Resolves statements via expression substitution.
- Once all operations are cleared, the statement is completed.
- For example:
 $6 / 2 * (1 + 2)$
 $6 / 2 * 3$
 $3 * 3$
 9
- Another example:

Statement Resolution

- C++ Resolves statements via expression substitution.
- Once all operations are cleared, the statement is completed.
- For example:
 $6/2 * (1+2)$
 $6/2 * 3$
 $3 * 3$
 9
- Another example:
`cout << 2+2 << endl`

Statement Resolution

- C++ Resolves statements via expression substitution.
- Once all operations are cleared, the statement is completed.

- For example:

$6/2 * (1+2)$

$6/2 * 3$

$3 * 3$

9

- Another example:

```
cout << 2+2 << endl
```

```
cout << 4 << endl
```

Statement Resolution

- C++ Resolves statements via expression substitution.
- Once all operations are cleared, the statement is completed.

- For example:

$6/2 * (1+2)$

$6/2 * 3$

$3 * 3$

9

- Another example:

```
cout << 2+2 << endl
```

```
cout << 4 << endl
```

```
cout << endl
```


Statement Resolution

- C++ Resolves statements via expression substitution.
- Once all operations are cleared, the statement is completed.

- For example:

$6/2 * (1+2)$

$6/2 * 3$

$3 * 3$

9

- Another example:

```
cout << 2+2 << endl
```

```
cout << 4 << endl
```

```
cout << endl
```

```
cout
```

Outline

- 1 Types
- 2 Arithmetic Operators
- 3 Programming With Operators

The Overall Process

- 1 Write (in English) the steps to perform the program.

The Overall Process

- 1 Write (in English) the steps to perform the program.
- 2 Write any formulae needed.

The Overall Process

- 1 Write (in English) the steps to perform the program.
- 2 Write any formulae needed.
- 3 Identify variables and constants.

The Overall Process

- 1 Write (in English) the steps to perform the program.
- 2 Write any formulae needed.
- 3 Identify variables and constants.
- 4 Decide on variable and constant types.

The Overall Process

- 1 Write (in English) the steps to perform the program.
- 2 Write any formulae needed.
- 3 Identify variables and constants.
- 4 Decide on variable and constant types.
- 5 Start with the boilerplate program.

The Overall Process

- 1 Write (in English) the steps to perform the program.
- 2 Write any formulae needed.
- 3 Identify variables and constants.
- 4 Decide on variable and constant types.
- 5 Start with the boilerplate program.
- 6 Declare variables and constants.

The Overall Process

- 1 Write (in English) the steps to perform the program.
- 2 Write any formulae needed.
- 3 Identify variables and constants.
- 4 Decide on variable and constant types.
- 5 Start with the boilerplate program.
- 6 Declare variables and constants.
- 7 Write code to perform the needed operations.

Lab Activity: Calculate Circle Area (Steps 1-3)

Problem: Write a program to calculate the area of a circle.

Lab Activity: Calculate Circle Area (Steps 1-3)

Problem: Write a program to calculate the area of a circle.
1.) Write steps in English

Lab Activity: Calculate Circle Area (Steps 1-3)

Problem: Write a program to calculate the area of a circle.

1.) Write steps in English

- 1 get the radius of the circle

Lab Activity: Calculate Circle Area (Steps 1-3)

Problem: Write a program to calculate the area of a circle.

1.) Write steps in English

- 1 get the radius of the circle
- 2 calculate the area

Lab Activity: Calculate Circle Area (Steps 1-3)

Problem: Write a program to calculate the area of a circle.

1.) Write steps in English

- 1 get the radius of the circle
- 2 calculate the area
- 3 print the results

Lab Activity: Calculate Circle Area (Steps 1-3)

Problem: Write a program to calculate the area of a circle.

1.) Write steps in English

- 1 get the radius of the circle
- 2 calculate the area
- 3 print the results

2.) Write any formulae needed

Lab Activity: Calculate Circle Area (Steps 1-3)

Problem: Write a program to calculate the area of a circle.

1.) Write steps in English

- 1 get the radius of the circle
- 2 calculate the area
- 3 print the results

2.) Write any formulae needed

$$a = \pi r^2$$

Lab Activity: Calculate Circle Area (Steps 1-3)

Problem: Write a program to calculate the area of a circle.

1.) Write steps in English

- 1 get the radius of the circle
- 2 calculate the area
- 3 print the results

2.) Write any formulae needed

$$a = \pi r^2$$

3.) Identify variables and constants.

Lab Activity: Calculate Circle Area (Steps 1-3)

Problem: Write a program to calculate the area of a circle.

1.) Write steps in English

- 1 get the radius of the circle
- 2 calculate the area
- 3 print the results

2.) Write any formulae needed

$$a = \pi r^2$$

3.) Identify variables and constants.

PI, r, a

Lab Activity: Calculate Circle Area (Steps 4-5)

4.) Decide on variable and constant types

Lab Activity: Calculate Circle Area (Steps 4-5)

4.) Decide on variable and constant types

1 `PI: double`

Lab Activity: Calculate Circle Area (Steps 4-5)

4.) Decide on variable and constant types

- 1 `PI: double`
- 2 `r: double`

Lab Activity: Calculate Circle Area (Steps 4-5)

4.) Decide on variable and constant types

- 1 `PI: double`
- 2 `r: double`
- 3 `a: double`

Lab Activity: Calculate Circle Area (Steps 4-5)

4.) Decide on variable and constant types

- ❶ `PI: double`
- ❷ `r: double`
- ❸ `a: double`

5.) Start with the boilerplate program

Lab Activity: Calculate Circle Area (Steps 4-5)

4.) Decide on variable and constant types

- ❶ `PI: double`
- ❷ `r: double`
- ❸ `a: double`

5.) Start with the boilerplate program

- ❶ Make your `labs/week3` directory.

Lab Activity: Calculate Circle Area (Steps 4-5)

4.) Decide on variable and constant types

- 1 `PI: double`
- 2 `r: double`
- 3 `a: double`

5.) Start with the boilerplate program

- 1 Make your `labs/week3` directory.
- 2 Copy `boilerplate.cpp` to `labs/week3/circle.cpp`.

Lab Activity: Calculate Circle Area (Step 6)

6.) Declare variable and Constants

Add the following to the beginning of main:

```
const double PI=3.14159; //the ratio c/d for all circles
double r;                //radius of the circle
double a;                //The area of the circle
```

Lab Activity: Calculate Circle Area (Step 7)

7.) Write code to perform the needed operations

Leave a blank line after the declarations and add the following:

```
//get the radius of the circle
cout << "What is the radius of the circle? ";
cin >> r;

//calculate the area
a = PI * r * r;

//print the results
cout << "Area: " << a << endl;
```

Compile and test your program.

Some Notes on Style

- Code within a block should be indented.

Some Notes on Style

- Code within a block should be indented.
- Variable declarations should go at the top of a block.

Some Notes on Style

- Code within a block should be indented.
- Variable declarations should go at the top of a block.
- A blank line should follow the variable declarations.

Some Notes on Style

- Code within a block should be indented.
- Variable declarations should go at the top of a block.
- A blank line should follow the variable declarations.
- A blank line should separate related chunks of code.

Some Notes on Style

- Code within a block should be indented.
- Variable declarations should go at the top of a block.
- A blank line should follow the variable declarations.
- A blank line should separate related chunks of code.
- Each chunk of code should have a comment introducing it.

A Correctly Formatted main for circle.cpp

```
int main()
{
    const double PI=3.14159; //the ratio c/d for all circles
    double r;                //radius of the circle
    double a;                //The area of the circle

    //get the radius of the circle
    cout << "What is the radius of the circle? ";
    cin >> r;

    //calculate the area
    a = PI * r * r;

    //print the results
    cout << "Area: " << a << endl;
}
```

Challenge: Additional Circle Calculations

Challenge: Add computation of diameter and circumference to your circle program.

For example, given a radius of 3, your program should produce the following output:

```
Diameter: 6  
Circumference: 18.8495  
Area: 28.2743
```

Lab Activity: Quadratic Equation

Problem: Compute the quadratic equation for any set of coefficients.

Lab Activity: Quadratic Equation

Problem: Compute the quadratic equation for any set of coefficients.

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

Lab Activity: Quadratic Equation

Problem: Compute the quadratic equation for any set of coefficients.

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

Let's carry out the design process.

The `cmath` Library

- `cmath` includes functions like you would find on a scientific calculator.

The `cmath` Library

- `cmath` includes functions like you would find on a scientific calculator.
- One of these is `sqrt` which computes the square root of a number.

The `cmath` Library

- `cmath` includes functions like you would find on a scientific calculator.
- One of these is `sqrt` which computes the square root of a number.
- For the rest, please see
`https://en.cppreference.com/w/cpp/header/cmath.`

The `cmath` Library

- `cmath` includes functions like you would find on a scientific calculator.
- One of these is `sqrt` which computes the square root of a number.
- For the rest, please see
`https://en.cppreference.com/w/cpp/header/cmath.`
- To use these functions, you must add the following line underneath the `#include <iostream>`:
`#include <cmath>`

Quadratic Equation Variable Declarations

Add the following to the appropriate part of your main function.

```
double a, b, c;      //coefficients
double x1, x2;       //roots
double rhs;          //right hand side of the numerator
double divisor;      //the divisor
```

Quadratic Equation: Coefficients

Add the following at the appropriate space

```
//get the coefficients  
cout << "a=";  
cin >> a;  
cout << "b=";  
cin >> b;  
cout << "c=";  
cin >> c;
```

Quadratic Equation: Compute and Display

```
//compute the right hand side of the numerator  
rhs = sqrt(b*b - 4.0 * a * c);  
  
//compute the divisor  
divisor = 2.0 * a;  
  
//compute the roots  
x1 = (-b - rhs) / divisor;  
x2 = (-b + rhs) / divisor;  
  
//print the results  
cout << "The roots are: " << x1 << ", " << x2 << endl;
```

Finishing Up

- You should have the following files in `labs/week3`:

Finishing Up

- You should have the following files in `labs/week3`:
 - `circle.cpp`

Finishing Up

- You should have the following files in `labs/week3`:
 - `circle.cpp`
 - `quadratic.cpp`

Finishing Up

- You should have the following files in `labs/week3`:
 - `circle.cpp`
 - `quadratic.cpp`
- Make sure both programs work.

Finishing Up

- You should have the following files in `labs/week3`:
 - `circle.cpp`
 - `quadratic.cpp`
- Make sure both programs work.
- Add, Commit, and Push in git!