CS12 CH:1-4 Review

Program Structure, Data Types, and Math

Mr. Gullo

October 8, 2025

Learning Objectives

Consolidated Review of Chapters 1-4

- Understand the basic 5-part structure of a C++ program.

Learning Objectives

Consolidated Review of Chapters 1-4

- Understand the basic 5-part structure of a C++ program.
- Identify and use fundamental data types: `int`, `float`, `char`, and `bool`.

Learning Objectives

Consolidated Review of Chapters 1-4

- Understand the basic 5-part structure of a C++ program.
- Identify and use fundamental data types: `int`, `float`, `char`, and `bool`.
- Differentiate between integer and floating-point division.

Learning Objectives

Consolidated Review of Chapters 1-4

- Understand the basic 5-part structure of a C++ program.
- Identify and use fundamental data types: `int`, `float`, `char`, and `bool`.
- Differentiate between integer and floating-point division.
- Explain fundamental memory concepts: Bit and Byte.

Learning Objectives

Consolidated Review of Chapters 1-4

- Understand the basic 5-part structure of a C++ program.
- Identify and use fundamental data types: `int`, `float`, `char`, and `bool`.
- Differentiate between integer and floating-point division.
- Explain fundamental memory concepts: Bit and Byte.
- Use `cin` to get input from a user.

Learning Objectives

Consolidated Review of Chapters 1-4

- Understand the basic 5-part structure of a C++ program.
- Identify and use fundamental data types: `int`, `float`, `char`, and `bool`.
- Differentiate between integer and floating-point division.
- Explain fundamental memory concepts: Bit and Byte.
- Use `cin` to get input from a user.
- Include and use the `<cmath>` library for advanced math calculations.

The 5 Parts of a C++ Program

A Quick Refresher

1. Pre-processor Directives

```
#include <...>
```

The 5 Parts of a C++ Program

A Quick Refresher

1. Pre-processor Directives

   ```
   #include <...>
   ```

2. Constant Definitions

   ```
   const float pi = ...;
   ```

The 5 Parts of a C++ Program

A Quick Refresher

1. Pre-processor Directives

   ```
   #include <...>
   ```

2. Constant Definitions

   ```
   const float pi = ...;
   ```

3. Main Body Heading

   ```
   int main() {}
   ```

The 5 Parts of a C++ Program

A Quick Refresher

1. Pre-processor Directives

   ```
   #include <...>
   ```

2. Constant Definitions

   ```
   const float pi = ...;
   ```

3. Main Body Heading

   ```
   int main() {}
   ```

4. Variable Declarations

   ```
   float diameter = ...;
   ```

The 5 Parts of a C++ Program

A Quick Refresher

1. Pre-processor Directives
   ```
   #include <...>
   ```
2. Constant Definitions
   ```
   const float pi = ...;
   ```
3. Main Body Heading
   ```
   int main() {}
   ```
4. Variable Declarations
   ```
   float diameter = ...;
   ```
5. Main Body Statements
   ```
   cout << ...;
   ```

The 5 Parts of a C++ Program

A Quick Refresher

1. Pre-processor Directives
   `#include <...>`
2. Constant Definitions
   `const float pi = ...;`
3. Main Body Heading
   `int main() {}`
4. Variable Declarations
   `float diameter = ...;`
5. Main Body Statements
   `cout << ...;`

```
1   #include <iostream>
2   using namespace std;
3
4   const float pi = 3.14159;
5
6   int main()
7   {
8       float diameter = 13.5;
9
10      cout << "Area is "
11           << pi * (diameter/2) * (diameter/2)
12           << endl;
13
14      return 0;
15  }
```

Code Comments

For Human Eyes Only

Comments are text in your code that is completely ignored by the compiler. Use them to explain *why* your code does something.

Code Comments

For Human Eyes Only

Comments are text in your code that is completely ignored by the compiler. Use them to explain *why* your code does something.

Single-Line Comments

Start with //. The compiler ignores everything to the end of the line.

```
// Calculate the area
float area = l * w;
```

Multi-Line Comments

Start with /* and end with */. Can span multiple lines.

```
/* This function calculates
   the distance between two
   points in a 2D plane. */
```

Fundamental Data Types

`int`

Stores positive and negative whole numbers.

Fundamental Data Types

`int`

Stores positive and negative whole numbers.

- Examples: -5, 0, 100

Fundamental Data Types

`int`

Stores positive and negative whole numbers.

- Examples: `-5`, `0`, `100`
- Operations: `+`, `-`, `*`, `/`, `%`

Fundamental Data Types

`int`

Stores positive and negative whole numbers.

- Examples: `-5`, `0`, `100`
- Operations: `+`, `-`, `*`, `/`, `%`

`char`

Stores a single character. Internally, it's a number (ASCII).

Fundamental Data Types

`int`

Stores positive and negative whole numbers.

- Examples: `-5`, `0`, `100`
- Operations: `+`, `-`, `*`, `/`, `%`

`char`

Stores a single character. Internally, it's a number (ASCII).

- Examples: `'A'`, `'?'`, `'\n'`

Fundamental Data Types

`int`

Stores positive and negative whole numbers.

- Examples: `-5`, `0`, `100`
- Operations: `+`, `-`, `*`, `/`, `%`

`float`

Stores numbers with decimal points.

`char`

Stores a single character. Internally, it's a number (ASCII).

- Examples: `'A'`, `'?'`, `'\n'`

Fundamental Data Types

`int`

Stores positive and negative whole numbers.

- Examples: `-5`, `0`, `100`
- Operations: `+`, `-`, `*`, `/`, `%`

`float`

Stores numbers with decimal points.

- Examples: `3.14`, `-0.05`

`char`

Stores a single character. Internally, it's a number (ASCII).

- Examples: `'A'`, `'?'`, `'\n'`

Fundamental Data Types

`int`

Stores positive and negative whole numbers.

- Examples: `-5`, `0`, `100`
- Operations: `+`, `-`, `*`, `/`, `%`

`float`

Stores numbers with decimal points.

- Examples: `3.14`, `-0.05`
- No modulo `%` operator.

`char`

Stores a single character. Internally, it's a number (ASCII).

- Examples: `'A'`, `'?'`, `'\n'`

Fundamental Data Types

`int`

Stores positive and negative whole numbers.

- Examples: `-5`, `0`, `100`
- Operations: `+`, `-`, `*`, `/`, `%`

`float`

Stores numbers with decimal points.

- Examples: `3.14`, `-0.05`
- No modulo `%` operator.

`char`

Stores a single character. Internally, it's a number (ASCII).

- Examples: `'A'`, `'?'`, `'\n'`

`bool`

Stores logical values: true or false.

Fundamental Data Types

`int`

Stores positive and negative whole numbers.

- Examples: `-5`, `0`, `100`
- Operations: `+`, `-`, `*`, `/`, `%`

`float`

Stores numbers with decimal points.

- Examples: `3.14`, `-0.05`
- No modulo `%` operator.

`char`

Stores a single character. Internally, it's a number (ASCII).

- Examples: `'A'`, `'?'`, `'\n'`

`bool`

Stores logical values: true or false.

- Internally: `true` is 1, `false` is 0.

Essential Concept: Integer vs. Float Division

This is one of the most common sources of bugs for new programmers!

Essential Concept: Integer vs. Float Division

This is one of the most common sources of bugs for new programmers!

- **Integer Division**: If both numbers are integers, the result is an integer. The decimal part is truncated (cut off).

Essential Concept: Integer vs. Float Division

This is one of the most common sources of bugs for new programmers!

- **Integer Division**: If both numbers are integers, the result is an integer. The decimal part is truncated (cut off).
  - `5 / 4` $\rightarrow$ 1

Essential Concept: Integer vs. Float Division

This is one of the most common sources of bugs for new programmers!

- **Integer Division**: If both numbers are integers, the result is an integer. The decimal part is truncated (cut off).
  - `5 / 4` $\rightarrow$ 1
- **Floating-Point Division**: If at least one number is a float, the result is a float, preserving the decimal.

Essential Concept: Integer vs. Float Division

This is one of the most common sources of bugs for new programmers!

- **Integer Division**: If both numbers are integers, the result is an integer. The decimal part is truncated (cut off).
  - `5 / 4` → `1`
- **Floating-Point Division**: If at least one number is a float, the result is a float, preserving the decimal.
  - `5.0 / 4` → `1.25`

Essential Concept: Integer vs. Float Division

This is one of the most common sources of bugs for new programmers!

- **Integer Division**: If both numbers are integers, the result is an integer. The decimal part is truncated (cut off).
  - `5 / 4` → `1`
- **Floating-Point Division**: If at least one number is a float, the result is a float, preserving the decimal.
  - `5.0 / 4` → `1.25`
  - `5 / 4.0` → `1.25`

Essential Concept: Integer vs. Float Division

This is one of the most common sources of bugs for new programmers!

- **Integer Division**: If both numbers are integers, the result is an integer. The decimal part is truncated (cut off).
  - `5 / 4` → `1`
- **Floating-Point Division**: If at least one number is a float, the result is a float, preserving the decimal.
  - `5.0 / 4` → `1.25`
  - `5 / 4.0` → `1.25`

Key Rule

The division type is determined *before* the result is assigned to a variable.

Context: Visualizing Memory

All data types—integers, characters, floats—are stored in the computer's memory as sequences of binary digits (bits).

Context: Visualizing Memory

All data types—integers, characters, floats—are stored in the computer's memory as sequences of binary digits (bits).
To understand how much space they take, we need to understand the basic units of memory: bits and bytes.

Visualization: 8 Bits in 1 Byte

[Diagram showing 8 individual squares (bits) labeled 0 or 1, grouped together into a larger rectangle labeled "1 Byte".]

Context: Visualizing Data Type Sizes

Different data types require different amounts of memory.

Context: Visualizing Data Type Sizes

Different data types require different amounts of memory.
The `sizeof()` operator in C++ tells us how many bytes a type uses.

Context: Visualizing Data Type Sizes

Different data types require different amounts of memory.
The `sizeof()` operator in C++ tells us how many bytes a type uses.
The next slide visualizes the typical sizes on a modern computer.

Visualization: Typical Data Type Sizes

[A bar chart showing the relative sizes of data types. X-axis: bool, char, int, float. Y-axis: Size in Bytes. Bars should show bool=1, char=1, int=4, float=4.]

Getting User Input with `cin`

- To make programs interactive, we read input from the user with `cin`.

Getting User Input with `cin`

- To make programs interactive, we read input from the user with `cin`.
- `cin` is part of the `<iostream>` library.

Getting User Input with `cin`

- To make programs interactive, we read input from the user with `cin`.
- `cin` is part of the `<iostream>` library.
- It uses the extraction operator >> to "pull" data from the keyboard into a variable.

Getting User Input with `cin`

- To make programs interactive, we read input from the user with `cin`.
- `cin` is part of the `<iostream>` library.
- It uses the extraction operator >> to "pull" data from the keyboard into a variable.

Example: Reading an integer

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

int main() {
    int age; // Variable to store input

    cout << "Please enter your age: "; // Prompt
    cin >> age; // Read keyboard input into 'age'

    cout << "You are " << age << " years old." << endl;

    return 0;
}
```

The <cmath> Library

- For advanced math, C++ provides the <cmath> library.

The <cmath> Library

- For advanced math, C++ provides the <cmath> library.
- You must include it at the top of your program to use its functions:

```
#include <cmath>
```

The <cmath> Library

- For advanced math, C++ provides the <cmath> library.
- You must include it at the top of your program to use its functions:

```
#include <cmath>
```

Common Functions

- `pow(base, exp)`: Calculates base to the power of exp. (x^y)

The <cmath> Library

- For advanced math, C++ provides the <cmath> library.
- You must include it at the top of your program to use its functions:

```
#include <cmath>
```

Common Functions

- `pow(base, exp)`: Calculates base to the power of exp. (x^y)
- `sqrt(x)`: Calculates the square root of x. ($\sqrt{x}$)

The <cmath> Library

- For advanced math, C++ provides the <cmath> library.
- You must include it at the top of your program to use its functions:

```
#include <cmath>
```

Common Functions

- `pow(base, exp)`: Calculates base to the power of exp. (x^y)
- `sqrt(x)`: Calculates the square root of x. ($\sqrt{x}$)
- `exp(x)`: Calculates the exponential function. (e^x)

Essential Equations Review

Distance Between Two Points

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Essential Equations Review

Distance Between Two Points

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Slope of a Line

$$\text{slope} = \frac{y_2 - y_1}{x_2 - x_1}$$

Essential Equations Review

Distance Between Two Points

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Slope of a Line

$$\text{slope} = \frac{y_2 - y_1}{x_2 - x_1}$$

Radioactive Decay

$$\text{remaining} = (\text{original}) \times e^{-0.00012t}$$

Exercise 1: Combined Arithmetic

Exercise File: `review_arithmetic.cpp`

Exercise 1: Combined Arithmetic

Exercise File: `review_arithmetic.cpp`

`Objective: Test your understanding of integer vs. float division by calculating an average.`

Exercise 1: Combined Arithmetic

Exercise File: `review_arithmetic.cpp`

Objective: Test your understanding of integer vs. float division by calculating an average.

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

int main() {
    int test1, test2, test3;

    // TODO 1: Prompt the user to enter three integer test scores.

    // TODO 2: Read the three scores into the variables test1, test2, and test3.

    // TODO 3: Calculate the average of the three scores.
    // BE CAREFUL! The result should be a float. How do you avoid
    // integer division here?

    // TODO 4: Print the average score to the console.

    return 0;
}
```

Exercise 2: Distance Formula

Exercise File: `review_distance.cpp`

Exercise 2: Distance Formula

Exercise File: `review_distance.cpp`

`Objective: Combine user input, floating-point math, and the <cmath> library.`

Exercise 2: Distance Formula

Exercise File: review_distance.cpp

Objective: Combine user input, floating-point math, and the <cmath> library.

```cpp
#include <iostream>
#include <cmath> // Don't forget this!
using namespace std;

int main() {
    float x1, y1, x2, y2;
    float distance;

    // TODO 1: Prompt the user for the coordinates of two points (x1, y1) and
    ↪  (x2, y2).

    // TODO 2: Read the four float values from the user.

    // TODO 3: Calculate the distance using the formula.
    // HINT: You will need pow() for squaring and sqrt() for the root.

    // TODO 4: Print the calculated distance.

    return 0;
}
```

Exercise 3: ASCII Character Math

Exercise File: `review_char_math.cpp`

Exercise 3: ASCII Character Math

Exercise File: `review_char_math.cpp`

`Objective: Reinforce that characters are numbers by converting character case.`

Exercise 3: ASCII Character Math

Exercise File: review_char_math.cpp

Objective: Reinforce that characters are numbers by converting character case.

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

int main() {
    char uppercaseChar;
    char lowercaseChar;

    // TODO 1: Prompt the user to enter a single uppercase letter.

    // TODO 2: Read the character into the 'uppercaseChar' variable.

    // TODO 3: Calculate the corresponding lowercase letter.
    // HINT: The ASCII value for 'a' is 32 greater than 'A'.
    // Perform an addition operation on the char variable.

    // TODO 4: Print the original uppercase letter and the new lowercase letter.

    return 0;
}
```

Summary: Key Takeaways

- **Structure is Key**: All C++ programs follow a predictable 5-part structure, which makes them easier to read and write.

Summary: Key Takeaways

- **Structure is Key**: All C++ programs follow a predictable 5-part structure, which makes them easier to read and write.
- **Data Types Matter**: The type of a variable (`int`, `float`, etc.) determines what it can store and how it behaves in calculations, especially division.

Summary: Key Takeaways

- **Structure is Key**: All C++ programs follow a predictable 5-part structure, which makes them easier to read and write.
- **Data Types Matter**: The type of a variable (`int`, `float`, etc.) determines what it can store and how it behaves in calculations, especially division.
- **Memory is Finite**: Data is stored in memory as bits and bytes. `sizeof()` shows us that different types have different memory footprints.

Summary: Key Takeaways

- **Structure is Key**: All C++ programs follow a predictable 5-part structure, which makes them easier to read and write.
- **Data Types Matter**: The type of a variable (`int`, `float`, etc.) determines what it can store and how it behaves in calculations, especially division.
- **Memory is Finite**: Data is stored in memory as bits and bytes. `sizeof()` shows us that different types have different memory footprints.
- **Programs are Interactive**: `cin` is our tool for getting input from the user, making our programs dynamic.

Summary: Key Takeaways

- **Structure is Key**: All C++ programs follow a predictable 5-part structure, which makes them easier to read and write.
- **Data Types Matter**: The type of a variable (`int`, `float`, etc.) determines what it can store and how it behaves in calculations, especially division.
- **Memory is Finite**: Data is stored in memory as bits and bytes. `sizeof()` shows us that different types have different memory footprints.
- **Programs are Interactive**: `cin` is our tool for getting input from the user, making our programs dynamic.
- **Libraries Extend Power**: We don't have to reinvent the wheel. Libraries like `<cmath>` provide powerful, pre-built functions to solve complex problems.