

CMPT 1109

Programming I

Shahriar Khosravi, Ph.D.

Lecture 2

Plan for Today

- The **cin** Object
- Mathematical Expressions
- Data Type Conversion
- Overflow and Underflow
- Type Casting
- Multiple Assignment and Combined Assignment
- Math Library Functions
- In-Class Exercise

The cin Object

The `cin` Object



*Pronounced “see-in” for
console input.*

The `cin` Object

- The **`cin`** object is the standard input object. It reads input from the console (or keyboard).
- Just like the **`cout`** object, the **`cin`** object requires the **`<iostream>`** file.

```
cout << "What is the length of the rectangle? ";  
cin >> length;
```

- The **`>>`** symbol is the **stream extraction operator**.
- This operator gets characters from the stream object on its left and stores them in the variable whose name appears on its right.
- In this example, characters are taken from the **`cin`** object (which gets them from the keyboard) and are stored in the `length` variable.
- The **`cin`** object automatically converts data to the type that matches the variable.
- The **`cin`** object causes a program to wait until data is typed at the keyboard and the Enter key is pressed. **No other lines in the program will be executed until `cin` gets its input.**

Example

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

int main()
{
    int length, width, area;

    cout << "This program calculates the area of a ";
    cout << "rectangle.\n";
    cout << "What is the length of the rectangle? ";
    cin >> length;
    cout << "What is the width of the rectangle? ";
    cin >> width;
    area = length * width;
    cout << "The area of the rectangle is " << area << ".\n";
    return 0;
}
```


Entering Multiple Values

- The **cin** object may be used to gather multiple values at once.
cin >> height >> width;
- Multiple values from keyboard must be separated by spaces (or the Enter key).
- Order is important: first value entered goes to first variable, etc.

```
// cin can read multiple values
// of different data types.
#include <iostream>
using namespace std;

int main()
{
    int whole;
    double fractional;
    char letter;

    cout << "Enter an integer, a double, and a character: ";
    cin >> whole >> fractional >> letter;
    cout << "Whole: " << whole << endl;
    cout << "Fractional: " << fractional << endl;
    cout << "Letter: " << letter << endl;
    return 0;
}
```

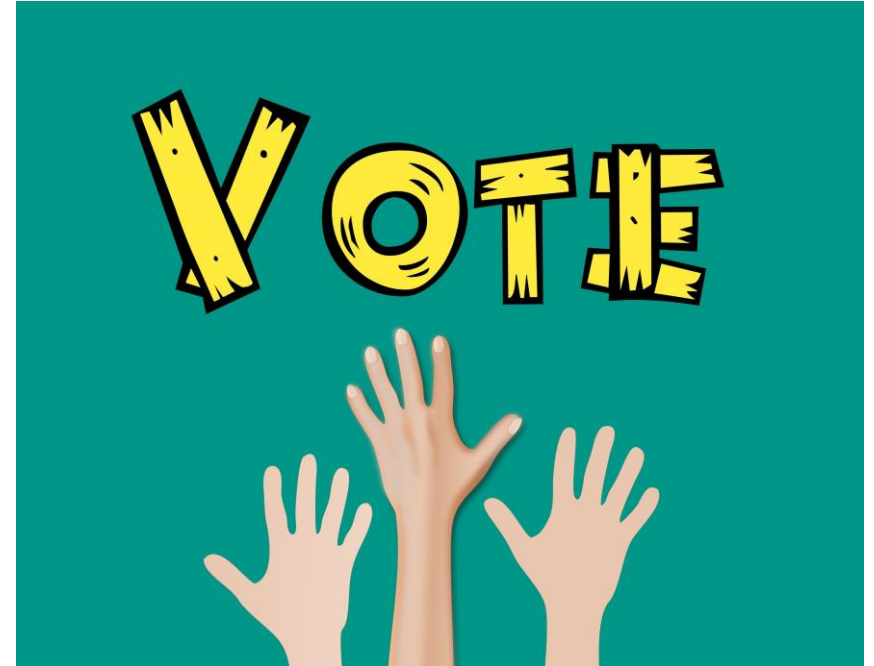
Poll 1 (Extra Credit)

Assume `value` is an `int` variable. If the user enters 3.14 in response, what will be stored in `value`?

```
cin >> value;
```

- a) 3.14
- b) 3
- c) Nothing, this leads to an error.

Please use the “Poll” window to participate for extra credit! One answer only please!





DOUGLAS COLLEGE

Mathematical Expressions

Mathematical Expressions

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

```
area = 2 * PI * radius;
```

```
result = sizeof(int);
```

```
cout << "border is: " << 2 * (1 + w);
```

Operator Precedence (Order of Operations)

- Consider the following statement:

outcome = 12 + 6 / 3;

- What value will be stored in **outcome**?



Operator Precedence (Order of Operations)

- Consider the following statement:

outcome = 12 + 6 / 3;

- What value will be stored in **outcome**?
- The answer is **14** because the division operator has higher **precedence** than the addition operator.
- Mathematical expressions are evaluated from left to right.**
- When two operators share an operand, the operator with the highest precedence works first.
- Multiplication and division have higher **precedence** than addition and subtraction.

Precedence Arithmetic Operators

1. $-$ (unary negation operator)
2. $*$ / $\%$
3. $+$ $-$

Expression	Value
$5 + 2 * 4$	13
$10 / 2 - 3$	2
$8 + 12 * 2 - 4$	28
$4 + 17 \% 2 - 1$	4
$6 - 3 * 2 + 7 - 1$	6

Operator Associativity

- An operator's **associativity** is either **left-to-right**, or **right-to-left**.
- If two operators sharing an operand have the same precedence, they work according to their associativity.

Operator	Associativity
(unary negation) -	Right-to-left
* / %	Left-to-right
+ -	Left-to-right

- For example, in this expression, $5 - 3$ is evaluated first, then the result is added to 2:
 $5 - 3 + 2;$
- Therefore, the expression is equivalent to $(5 - 3) + 2;$

Grouping with Parentheses

- Parts of a mathematical expression may be grouped with parentheses to force some operations to be performed before others, as shown below.

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

Algebraic Expressions

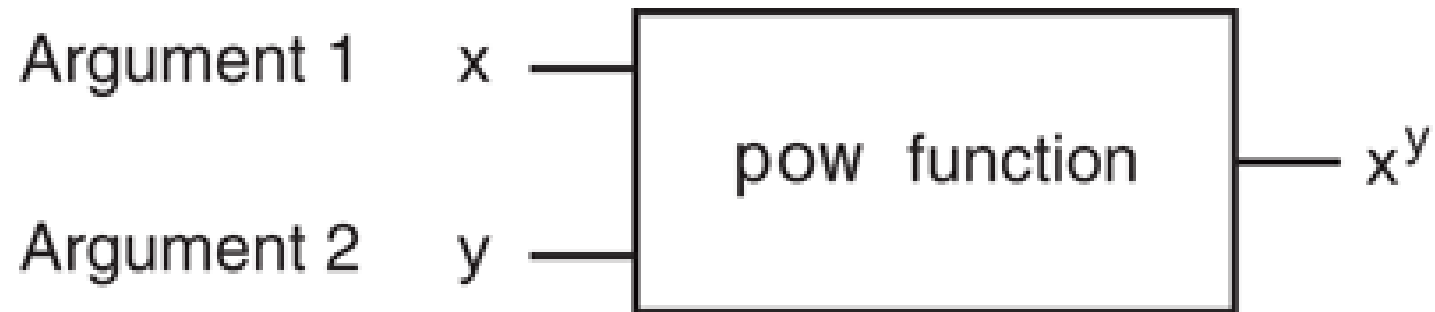
Algebraic Expression	C++ Expression
$y = \frac{3x}{2}$	<code>y = x / 2 * 3;</code>
$z = 3bc + 4$	<code>z = 3 * b * c + 4;</code>
$a = \frac{3x + 2}{4a - 1}$	<code>a = (3 * x + 2) / (4 * a - 1)</code>

- Unlike many programming languages, **C++ does not have an exponent operator**. Raising a number to a power requires the use of a **library function**.
- The C++ library is a collection of specialized functions.
- One of the library functions is called **pow()**, and its purpose is to raise a number to a power:
`area = pow(4.0, 2.0);`
- This C++ statement is equivalent to $area = 4^2$.

The pow() Function

area = $\xleftarrow{\text{16.0 return value}}$ pow(4.0, 2.0) ;

arguments



The pow() function can be viewed as a “black box” that turns two arguments into an output value.

Example

```
// This program calculates the area of a circle.
#include <iostream>
#include <cmath>    // needed for pow function
using namespace std;

int main()
{
    const double PI = 3.14159;
    double area, radius;

    cout << "This program calculates the area of a circle.\n";
    cout << "What is the radius of the circle? ";
    cin >> radius;
    area = PI * pow(radius, 2.0);
    cout << "The area is " << area << endl;
    return 0;
}
```



DOUGLAS COLLEGE

Type Conversion

Type Conversion

- When an operator's operands are of different data types, **C++ will automatically convert them to the same data type.**
- This can affect the results of mathematical expressions.
- If an **int** is multiplied by a **float**, what data type will the result be?
- What if a **double** is divided by an **unsigned int**?
- Is there any way of predicting what will happen in these instances?



Type Conversion

- When an operator's operands are of different data types, **C++ will automatically convert them to the same data type.**
- This can affect the results of mathematical expressions.
- If an **int** is multiplied by a **float**, what data type will the result be?
- What if a **double** is divided by an **unsigned int**?
- Is there any way of predicting what will happen in these instances?
- Fortunately, follows a set of rules when performing mathematical operations on variables of different data types.
- In C++, data types are ranked.
- One data type outranks another if it can hold a larger number.
- For example, a **float** outranks an **int**.

Hierarchy of Types

- Highest
 1. long double
 2. double
 3. float
 4. unsigned long long int
 5. long long int
 6. unsigned long int
 7. long int
 8. unsigned int
 9. int
- Lowest
- An exception to the ranking is when an **int** and a **long** are the same size. In that case, an **unsigned int** outranks **long** because it can hold a higher value.

Type Conversion

- When C++ is working with an operator, it strives to convert the operands to the same type.
- This automatic conversion is known as **type coercion**.
- When a value is converted to a higher data type, it is **promoted**. To **demote** a value means to convert it to a lower data type.
- Coercion rules:
 - **char, short, unsigned short** automatically promoted to **int**.
 - When operating on values of different data types, **the lower one is promoted to the type of the higher one**.
 - When using the assignment operator =, **the type of expression on right will be converted to type of variable on left**.
- Note that **char, short, and unsigned short** do **not** appear in the rankings because if they are used in a mathematical expression, they are automatically promoted to an **int**.

Integer Division

- When we divide an **int** by another **int** in C++, the result is always an **int**.
- If there is a remainder, it will be discarded.
- For example, in the following code, parts is assigned the value **2.0**:

```
double parts;  
parts = 15 / 6;
```

- For a division operation to return a floating-point value, at least one of the operands must be of a floating-point data type:

```
double parts;  
parts = 15.0 / 6;
```

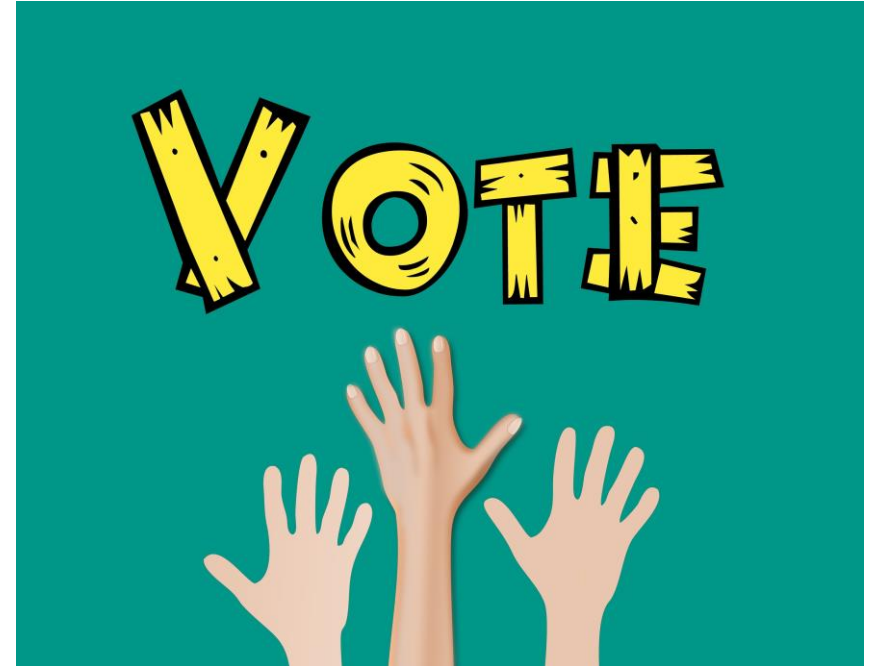
Poll 1 (Extra Credit)

What will the following program print?

```
#include <iostream>
using namespace std;
int main()
{
    short number = 65536;
    cout << number << endl;
    return 0;
}
```

- a) 0
- b) 65536
- c) I feel like this is a trick question.

Please use the “Poll” window to participate for extra credit! One answer only please!



Overflow and Underflow

- **Overflow** occurs when assigning a value that is **too large** to be held in a variable.
- **Underflow** occurs when assigning a value that is **too small** to be held in a variable.
- When either of these occurs, the variable will receive a 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.

Overflow and Underflow

- **Overflow** occurs when assigning a value that is **too large** to be held in a variable.
- **Underflow** occurs when assigning a value that is **too small** to be held in a variable.
- When either of these occurs, the variable will receive a 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.

```
#include <iostream>
using namespace std;
int main()
{
    unsigned short number = -1;
    cout << number << endl;
    return 0;
}
```

← This will **underflow**!

Type Casting

Type Casting

- A **type cast expression** allows us to manually promote or demote a value.
- The general format of a type cast expression is
static_cast<DataType>(Value);
- Here, **Value** is a variable or literal value that we wish to convert, and **DataType** is the data type to which we wish to convert **Value**.
- Here is an example of code that uses a type cast expression:

```
double number = 3.7;  
int val;  
val = static_cast<int>(number);
```
- Type cast expressions are useful in situations where C++ will not perform the desired conversion automatically.

Example

```
// This program uses a type cast to avoid integer division.
#include <iostream>
using namespace std;

int main()
{
    int books;           // Number of books to read
    int months;          // Number of months spent reading
    double perMonth;     // Average number of books per month

    cout << "How many books do you plan to read? ";
    cin >> books;
    cout << "How many months will it take you to read them? ";
    cin >> months;
    perMonth = static_cast<double>(books) / months;
    cout << "That is " << perMonth << " books per month.\n";
    return 0;
}
```

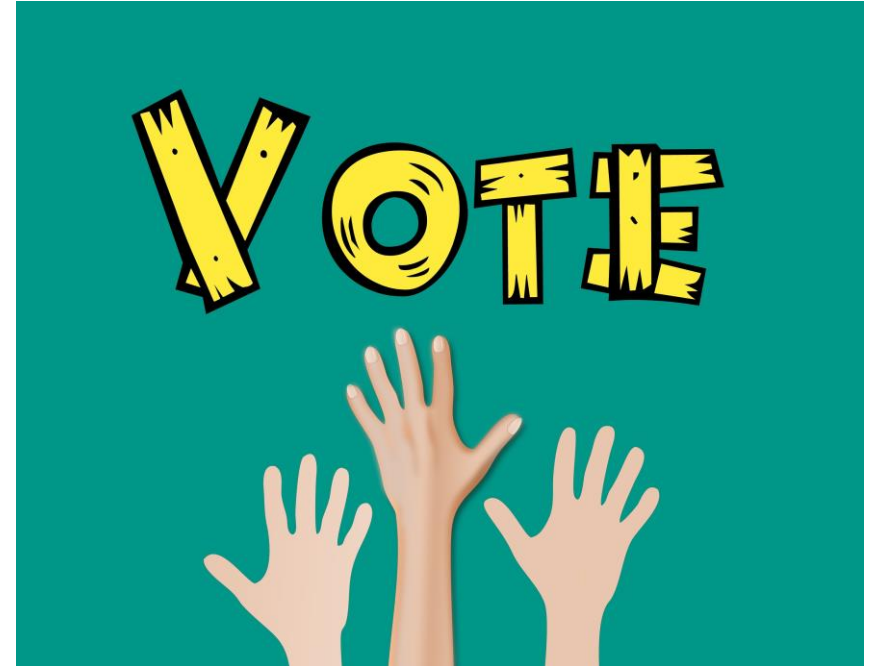
Poll 1 (Extra Credit)

Assuming books and months are integers, the following statement will result in integer division.

```
perMonth = static_cast<double>(books / months);
```

- a) Yes!
- b) No!

Please use the “Poll” window to participate for extra credit! One answer only please!

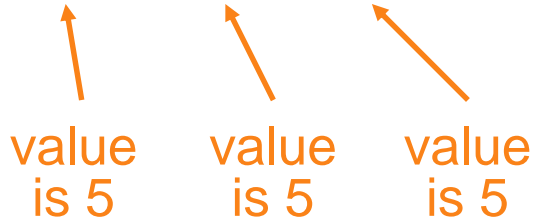


C-Style and Pre-standard Type Cast Expressions

- C-Style cast: data type name is provided within ()
 - `cout << ch << " is " << (int)ch;`
- Pre-standard C++ cast: value in ()
 - `cout << ch << " is " << int(ch);`
- Both are still supported in C++, although **static_cast** is preferred

Multiple Assignment and Combined Assignment

Multiple Assignment and Combine Assignment

- The assignment operator = can be used to assign a value to **multiple** variables:
 - `x = y = z = 5;`
- This operator associates from right to left:
 - `x = (y = (z = 5));`
- The following statement adds 1 to sum (**combined assignment**):
 - `sum = sum + 1;`

Examples

Statement	What It Does	Value of x After the Statement
$x = x + 4;$	Adds 4 to x	10
$x = x - 3;$	Subtracts 3 from x	3
$x = x * 10;$	Multiplies x by 10	60
$x = x / 2;$	Divides x by 2	3
$x = x \% 4$	Makes x the remainder of $x / 4$	2

Combined Assignment Operators

- The combined assignment operators provide a shorthand for these types of statements.
- The statement
 - `sum = sum + 1;`
- is equivalent to
 - `sum += 1;`

Operator	Example Usage	Equivalent to
<code>+=</code>	<code>x += 5;</code>	<code>x = x + 5;</code>
<code>-=</code>	<code>y -= 2;</code>	<code>y = y - 2;</code>
<code>*=</code>	<code>z *= 10;</code>	<code>z = z * 10;</code>
<code>/=</code>	<code>a /= b;</code>	<code>a = a / b;</code>
<code>%=</code>	<code>c %= 3;</code>	<code>c = c % 3;</code>

Formatting Output

Formatting Output

- The way a value is printed onto the console screen is called its **formatting**.
- The **cout** object provides ways to format data as it is being displayed.
- This affects the way data appears on the screen.
- The same data can be printed or displayed in several different ways.
- For example, all of the following numbers have the same value, although they look different:

720

720.0

720.00000000

7.2e+2

+720.0

Stream Manipulators

- **Stream manipulators** are used to control how an output field is displayed
- One stream manipulator is **setw(x)** which prints in a field at least **x** spaces wide.
- **setw()** only specifies the minimum number of positions in the print field. Any number larger than the minimum will cause **cout** to override the **setw()** value.
- **setw()** requires the **<iomanip>** library.
- **setw()** **only affects the next value displayed.**

```
// This program displays three rows of numbers.
#include <iostream>
#include <iomanip>      // Required for setw
using namespace std;

int main()
{
    int num1 = 2897, num2 = 5, num3 = 837,
        num4 = 34, num5 = 7, num6 = 1623,
        num7 = 390, num8 = 3456, num9 = 12;

    // Display the first row of numbers
    cout << setw(6) << num1 << setw(6)
         << num2 << setw(6) << num3 << endl;

    // Display the second row of numbers
    cout << setw(6) << num4 << setw(6)
         << num5 << setw(6) << num6 << endl;

    // Display the third row of numbers
    cout << setw(6) << num7 << setw(6)
         << num8 << setw(6) << num9 << endl;
    return 0;
}
```

Stream Manipulators

- Some stream manipulators affect values until changed again:
 - **fixed**: use decimal notation for floating-point values.
 - **setprecision(x)**: when used with **fixed**, print floating-point value using **x** digits after the decimal. Without **fixed**, print floating-point value using **x** significant digits.
 - **showpoint**: always print decimal for floating-point values.

```
double x = 123.4, y = 456.0;  
cout << setprecision(6) << showpoint << x << endl;  
cout << y << endl;
```

Stream Manipulator	Description
<code>setw(n)</code>	Establishes a print field of <i>n</i> spaces.
<code>fixed</code>	Displays floating-point numbers in fixed point notation.
<code>showpoint</code>	Causes a decimal point and trailing zeroes to be displayed, even if there is no fractional part.
<code>setprecision(n)</code>	Sets the precision of floating-point numbers.
<code>left</code>	Causes subsequent output to be left justified.
<code>right</code>	Causes subsequent output to be right justified.

Working with Characters and string Objects

Working with Characters and string Objects

- Although it is possible to use **cin** with the extraction >> operator to input **string** objects, it can cause problems.
- When **cin** reads input, it passes over and ignores any leading whitespace characters (spaces, tabs, or line breaks).
- Once it comes to the first nonblank character and starts reading, it stops reading when it gets to the next whitespace character.

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

int main()
{
    string name;
    string city;

    cout << "Please enter your name: ";
    cin >> name;
    cout << "Enter the city you live in: ";
    cin >> city;

    cout << "Hello, " << name << endl;
    cout << "You live in " << city << endl;
    return 0;
}
```

Working with Characters and string Objects

- Although it is possible to use `cin`

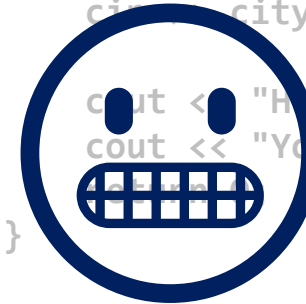
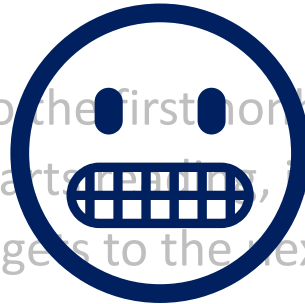
```
#include <iostream>
```

Microsoft Visual Studio Debug Console

```
Please enter your name: Shahriar Khosravi
Enter the city you live in: Hello, Shahriar
You live in Khosravi
```

characters (spaces, tabs, or line breaks).

- Once it comes to the first non-blank character and starts reading, it stops reading when it gets to the next whitespace character.



```
cout << "Enter the city you live in: ";
cin >> city;
```

```
cout << "Hello, " << name << endl;
cout << "You live in " << city << endl;
```

```
}
```

Working with Characters and string Objects

- To work around this problem, we can use a C++ function named **getline()**.
- The **getline()** function reads an entire line, including leading and embedded spaces, and stores it in a **string** object.
- The **getline()** function looks like the following, where **cin** is the input stream we are reading from and **inputLine** is the name of the **string** object receiving the input: **getline(cin, inputLine);**

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

int main()
{
    string name;
    string city;

    cout << "Please enter your name: ";
    getline(cin, name);
    cout << "Enter the city you live in: ";
    getline(cin, city);

    cout << "Hello, " << name << endl;
    cout << "You live in " << city << endl;
    return 0;
}
```

Inputting A Character

- Sometimes we want to read only a single character of input.
- For example, some programs display a menu of items for the user to choose from.
- We can use the **cin** object in these cases, but this is sometimes undesirable because **cin** passes over all leading whitespace, it is impossible to input just a blank or Enter with **cin >>**.
- The program will not continue past the **cin** statement until some character other than the spacebar, tab key, or Enter key has been pressed.
- As a result, programs that ask the user to "Press the Enter key to continue." cannot use the extraction operator **>>** along with **cin** to read only the pressing of the Enter key.
- In those situations, the **cin** object has a built-in function named **get()** that is helpful.

Example

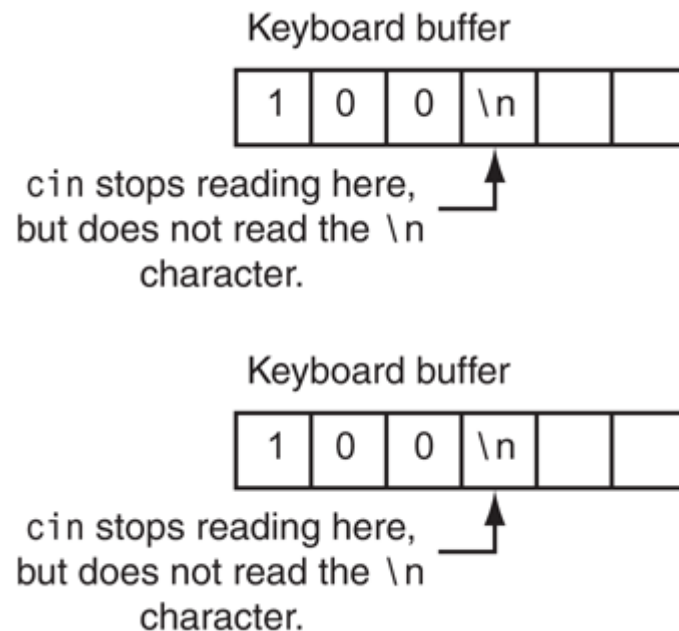
```
// This program demonstrates three ways
// to use cin.get() to pause a program
#include<iostream>
using namespace std;

int main()
{
    char ch;

    cout << "This program has paused. Press Enter to continue.";
    cin.get(ch);
    cout << "It has paused a second time. Please press Enter again.";
    ch = cin.get();
    cout << "It has paused a third time. Please press Enter again.";
    cin.get();
    cout << "Thank you!";
    return 0;
}
```

Mixing `cin >>` and `cin.get()`

- Mixing `cin >>` and `cin.get()` in the same program can cause input errors that are difficult to detect.
- The issue is illustrated below.



```
#include <iostream>
using namespace std;
int main()
{
    char ch;
    int number;

    cout << "Enter a number: ";
    cin >> number;
    cout << "Enter a character: ";
    ch = cin.get();
    cout << "Thank You!\n";
    return 0;
}
```

Using `cin.ignore()`

- To solve the problem previously described, we can use the member function **`ignore()`** from the **`cin`** object.
- The **`cin.ignore()`** function tells the `cin` object to skip one or more characters in the keyboard buffer.
- To skip over unneeded characters that are still in the keyboard buffer, use **`cin.ignore()`**:

```
// skip next char  
cin.ignore();
```

```
// skip the next 10 chars or until '\n'  
cin.ignore(10, '\n');
```

- The previous statement causes **`cin`** to skip the next **10** characters or until a newline is encountered, **whichever comes first**.

Example

```
#include <iostream>
using namespace std;
int main()
{
    char ch;
    int number;

    cout << "Enter a number: ";
    cin >> number;
    cin.ignore();
    cout << "Enter a character: ";
    ch = cin.get();
    cout << "Thank You!\n";
    return 0;
}
```

string Member Functions and Operators

- To find the length of a string:

```
string province = "Ontario";  
int size = province.length();
```

- To concatenate (join) multiple strings:

```
greeting2 = greeting1 + name1;  
greeting1 = greeting1 + name2;
```

- Or using the += combined assignment operator:

```
greeting1 += name2;
```

More Useful Math Library Functions

More Math Library Functions

- The following match library functions require the `<cmath>` header file to be included.
- Accept **double** as input, and return a **double** value.

Function	Example	Description
abs	<code>y = abs(x);</code>	Returns the absolute value of the argument. The argument and the return value are integers.
cos	<code>y = cos(x);</code>	Returns the cosine of the argument. The argument should be an angle expressed in radians. The return type and the argument are doubles.
exp	<code>y = exp(x);</code>	Computes the exponential function of the argument, which is x. The return type and the argument are doubles.
fmod	<code>y = fmod(x, z);</code>	Returns, as a double, the remainder of the first argument divided by the second argument. Works like the modulus operator, but the arguments are doubles. (The modulus operator only works with integers.) Take care not to pass zero as the second argument. Doing so would cause division by zero.
log	<code>y = log(x);</code>	Returns the natural logarithm of the argument. The return type and the argument are doubles.
log10	<code>y = log10(x);</code>	Returns the base-10 logarithm of the argument. The return type and the argument are doubles.
round	<code>y = round(x)</code>	The argument, x, can be a double, a float, or a long double. Returns the value of x rounded to the nearest whole number. For example, if x is 2.8, the function returns 3.0, or if x is 2.1, the function returns 2.0. The return type is the same as the type of the argument.
sin	<code>y = sin(x);</code>	Returns the sine of the argument. The argument should be an angle expressed in radians. The return type and the argument are doubles.
sqrt	<code>y = sqrt(x);</code>	Returns the square root of the argument. The return type and argument are doubles.
tan	<code>y = tan(x);</code>	Returns the tangent of the argument. The argument should be an angle expressed in radians. The return type and the argument are doubles.

Random Numbers

- Random numbers are useful in many applications, such as:
 - Games and simulations
 - Statistical analysis
 - Data encryption
- To generate random numbers in C++, we first need to **#include** the `<random>` library.
- **Example:** Generate a random integer in the range 0-100:

```
random_device myEngine;
uniform_int_distribution<int> randomInt(0, 100);
int number = randomInt(myEngine);
```

Creates a random number engine named `myEngine`

Creates a distribution object named `randomInt`

Generates a random `int` in the range 0-100 and assigns it to `number`

Example

```
// This program simulates rolling dice.
#include <iostream>
#include <random>
using namespace std;

int main()
{
    // Constants
    const int MIN = 1;    // Minimum dice value
    const int MAX = 6;    // Maximum dice value

    // Random number engine
    random_device engine;

    // Distribution object
    uniform_int_distribution<int> diceValue(MIN, MAX);

    cout << "Rolling the dice...\n";
    cout << diceValue(engine) << endl;
    cout << diceValue(engine) << endl;
    return 0;
}
```

In-Class Exercise

General Crates, Inc. builds custom-designed wooden crates. With materials and labor, it costs GCI \$0.23 per cubic foot to build a crate. In turn, they charge their customers \$0.50 per cubic foot for the crate. Write a C++ program that calculates the:

Volume (in cubic feet)

Cost

Customer price

Profit of any crate GCI builds



In-Class Exercise – Program Design

- The program must perform the following general steps:
- Step 1:
 - Ask the user to enter the dimensions of the crate.
- Step 2:
 - Calculate:
 - the crate's volume
 - the cost of building the crate
 - the customer's charge
 - the profit made
- Step 3:
 - Display the data calculated in Step 2.



Thank you.
DOUGLASCOLLEGE

DOUGLASCOLLEGE