CS 1002 Programming Fundamentals
Lecture #10 21 Sept 2022

Operators and Expressions

National University
Of Computer and Emerging Sciences

The \n Escape Sequence & endl Manipulator

- You can also use the \n escape sequence to start a new line of output. This will produce two lines of output:

```
cout << "Programming is\n";
cout << "fun!";
```

Notice that the `\n` is INSIDE the string.

```
cout << "Programming is" << endl;
cout << "fun!";
```

Same next line using endl manipulator

Common Escape Sequence

Escape Sequence	Name	Description
`\n`	Newline	Causes the cursor to go to the next line for subsequent printing.
`\t`	Horizontal tab	Causes the cursor to skip over to the next tab stop.
`\a`	Alarm	Causes the computer to beep.
`\b`	Backspace	Causes the cursor to back up, or move left one position.
`\r`	Return	Causes the cursor to go to the beginning of the current line, not the next line.
`\\`	Backslash	Causes a backslash to be printed.
`\'`	Single quote	Causes a single quotation mark to be printed.
`\"`	Double quote	Causes a double quotation mark to be printed.

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);
```

Arithmetic Operators

- Used for performing numeric calculations
- C++ has unary, binary, and ternary operators:
  - unary (1 operand) -5
  - binary (2 operands) 13 - 7
  - ternary (3 operands) exp1 ? exp2 : exp3

SYMBOL	OPERATION	EXAMPLE	VALUE OF ans
+	addition	`ans = 7 + 3;`	10
-	subtraction	`ans = 7 - 3;`	4
*	multiplication	`ans = 7 * 3;`	21
/	division	`ans = 7 / 3;`	2
%	modulus	`ans = 7 % 3;`	1

Order of Precedence

- All operations inside of () are evaluated first
- *, /, and % are at the same level of precedence and are evaluated next
- + and – have the same level of precedence and are evaluated last
- When operators are on the same level
  - Performed from left to right (associativity)

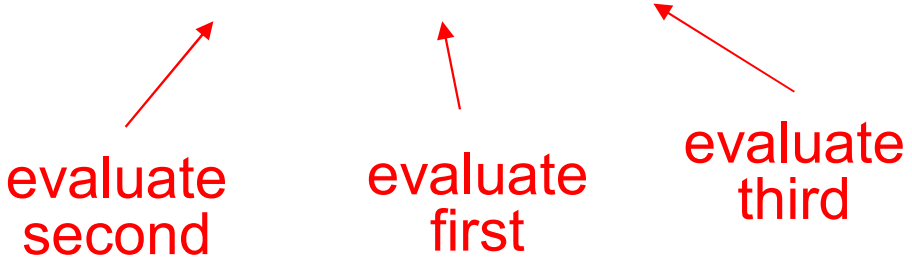
```
3 * 7 - 6 + 2 * 5 / 4 + 6 means
(((3 * 7) - 6) + ((2 * 5) / 4 )) + 6
```

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`



Order of Operations

Table 3-2 Some Simple Expressions and Their Values

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

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
```

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

A Closer Look at the / Operator

- `/` (division) operator performs integer division if both operands are integers

```
cout << 13 / 5;     // displays 2
cout << 91 / 7;     // displays 13
```

- If either operand is floating point, the result is floating point

```
cout << 13 / 5.0;   // displays 2.6
cout << 91.0 / 7;   // displays 13.0
```


- `/` (division) operator performs integer division if both operands are integers

```
cout << 13 / 5;     // displays 2
cout << 91 / 7;     // displays 13
```

- If either operand is floating point, the result is floating point

```
cout << 13 / 5.0;   // displays 2.6
cout << 91.0 / 7;   // displays 13.0
```

Expressions

- If all operands are integers
  - Expression is called an integral expression
    - Yields an integral result
    - Example: `2 + 3 * 5`
- If all operands are floating-point
  - Expression is called a floating-point expression
    - Yields a floating-point result
    - Example: `12.8 * 17.5 - 34.50`

Mixed Expressions

- Mixed expression:
  - Has operands of different data types
  - Contains integers and floating-point
- Examples of mixed expressions:

```
2 + 3.5
6  /  4 + 3.9
5.4  *  2 - 13.6 + 18  /  2
```

Mixed Expressions (continued)

- Evaluation rules:
  - If operator has same types of operands
    - Evaluated according to the type of the operands
  - If operator has both types of operands
    - Integer is changed to floating-point
    - Operator is evaluated
    - Result is floating-point
  - Entire expression is evaluated according to precedence rules

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:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

is written as

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

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

Named Constants

- Named constant (constant variable): variable whose content cannot be changed during program execution
- Used for representing constant values with descriptive names:

```
const double TAX_RATE = 0.0675;
const int NUM_STATES = 50;
```

- Often named in uppercase letters

Named Constants in Program 2-28

Program 2-28

```
1  // This program calculates the circumference of a circle.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     // Constants
8     const double PI = 3.14159;
9     const double DIAMETER = 10.0;
10
11    // Variable to hold the circumference
12    double circumference;
13
14    // Calculate the circumference.
15    circumference = PI * DIAMETER;
16
17    // Display the circumference.
18    cout << "The circumference is: " << circumference << endl;
19    return 0;
20 }
```

Program Output

```
The circumference is: 31.4159
```

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
```

Lowest:

```
int
```

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

- Rules of Type Coercion

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

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

Type Conversion (Casting)

- Implicit type coercion: when value of one type is automatically changed to another type

- Cast operator: provides explicit type conversion

```
static_cast<dataTypeName>(expression)
```

- 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 Conversion (continued)

EXAMPLE 2-9

Expression	Evaluates to
`static_cast<int>(7.9)`	`7`
`static_cast<int>(3.3)`	`3`
`static_cast<double>(25)`	`25.0`
`static_cast<double>(5+3)`	`= static_cast<double>(8) = 8.0`
`static_cast<double>(15) / 2`	`= 15.0 / 2` (because `static_cast<double>(15) = 15.0`) `= 15.0 / 2.0 = 7.5`
`static_cast<double>(15 / 2)`	`= static_cast<double>(7)` (because `15 / 2 = 7`) `= 7.0`
`static_cast<int>(7.8 + static_cast<double>(15) / 2)`	`= static_cast<int>(7.8 + 7.5)` `= static_cast<int>(15.3)` `= 15`
`static_cast<int>(7.8 + static_cast<double>(15 / 2))`	`= static_cast<int>(7.8 + 7.0)` `= static_cast<int>(14.8)` `= 14`

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 Pre-standard 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

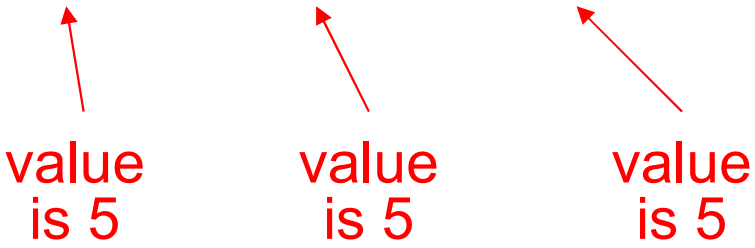
Multiple Assignment and Combined Assignment

- The = can be used to assign a value to multiple variables:

```
x = y = z = 5;
```

- Value of = is the value that is assigned
- Associates right to left:

```
x = (y = (z = 5));
```



Combined Assignment

- Look at the following statement:

```
sum = sum + 1;
```

This adds 1 to the variable **`sum`**.

Table 3-8 (Assume x = 6)

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

- The combined assignment operators provide a shorthand for these types of statements.
- The statement

```
sum = sum + 1;
```

is equivalent to

```
sum += 1;
```

Table 3-9

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

Math Functions with `<cmath>`

`abs(x)`	computes absolute value of x
`sqrt(x)`	computes square root of x, where x >=0
`pow(x,y)`	computes x^y
`ceil(x)`	nearest integer larger than x
`floor(x)`	nearest integer smaller than x
`exp(x)`	computes e^x
`log(x)`	computes ln x, where x >0
`log10(x)`	computes $\log_{10}x$, where x>0

Determining the Size of a Data Type

The `sizeof` operator gives the size of any data type or variable:

```
#include <iostream>
using namespace std;
int main() {
cout << "Size of char : " << sizeof(char) << endl;
cout << "Size of int : " << sizeof(int) << endl;
cout << "Size of expression 5 + 8 is : " << sizeof(5 + 8) << endl;
return 0;
}
```

```
Size of char : 1
Size of int : 4
Size of expression 5 + 8 is : 4
```

`string` Member Functions and Operators

- To find the length of a string:

```
string state = "Texas";
int size = state.length();
```

- To concatenate (join) multiple strings:

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

Or using the **+=** combined assignment operator:

```
greeting1 += name2;
```

Relational Operators

- Used to compare numbers to determine relative order
- Operators:

>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

Relational Expressions

- Boolean expressions – `true` or `false`
- Examples:

  `12 > 5` is `true`

  `7 <= 5` is `false`

  if `x` is 10, then

  `x == 10` is `true`,

  `x != 8` is `true`, and

  `x == 8` is `false`

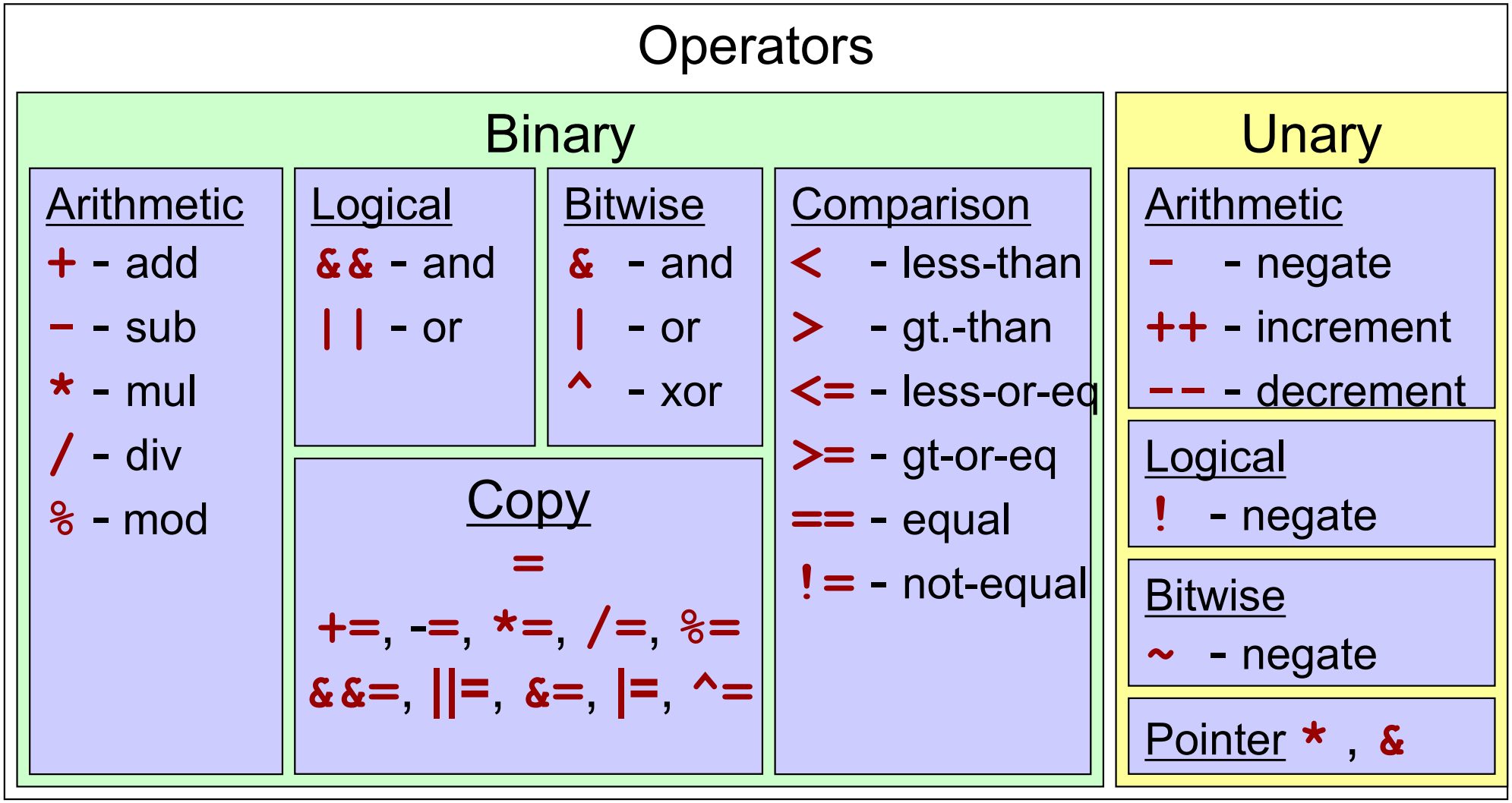
Can be assigned to a variable:

```
result = x <= y;
```

Assigns `0` for `false`, `1` for `true`

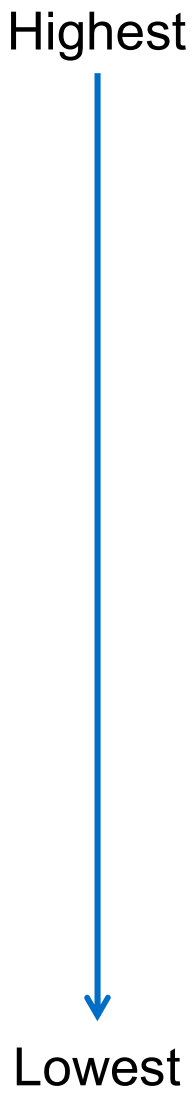
Do not confuse = and ==

C++ Operator Map



Precedence Chart

- ++, --, !, - (unary minus), + (unary plus)
- *, /, %
- + (addition), - (subtraction)
- <<, >>
- <, <=, >, >=
- ==, !=
- &&
- ||
- =



Unary Operators

- Negate: `-a` gives -9
- Logical-invert: `!a` gives 0
- `*` and `&` are pointer operations (discussed later)
- Increment: `++`
- Decrement: `--`

```
int a(9);
```

Increment & Decrement Operators

- Increment operator: increment variable by 1
- Decrement operator: decrement variable by 1
- Pre-increment: `++variable`
- Post-increment: `variable++`
- Pre-decrement: `--variable`
- Post-decrement: `variable--`

Increment & Decrement Operators (continued)

- `++count;` or `count++;` increments the value of `count` by `1`
- `--count;` or `count--;` decrements the value of `count` by
- If `x = 5;` and `y = ++x;`
  - After the second statement both `x` and `y` are `6`
- If `x = 5;` and `y = x++;`
  - After the second statement `y` is `5` and `x` is `6`

More Unary Expressions

```
int a(9), b;
```

- Pre-Increment :
  - `b = ++a;` b is 10 and a is 10
- Post-Increment:
  - `b = a++;` b is 9 and a is 10
- Pre-Decrement :
  - `b = --a;` b is 8 and a is 8
- Post-Decrement:
  - `b = a--;` b is 9 and a is 8

Hand Tracing a Program

- Hand trace a program: act as if you are the computer, executing a program:
  - step through and ‘execute’ each statement, one-by-one
  - record the contents of variables after statement execution, using a hand trace chart (table)
- Useful to locate logic or mathematical errors

Program with Hand Trace Chart

Program 3-26 **(with hand trace chart filled)**

```
1  // This program asks for three numbers, then
2  // displays the average of the numbers.
3  #include <iostream>
4  using namespace std;

5  int main()

6  {
7     double num1, num2, num3, avg;
8     cout << "Enter the first number: ";
9     cin >> num1;
10    cout << "Enter the second number: ";
11    cin >> num2;
12    cout << "Enter the third number: ";
13    cin >> num3;
14    avg = num1 + num2 + num3 / 3;
15    cout << "The average is " << avg << endl;
16    return 0;
17 }
```

num1	num2	num3	avg
?	?	?	?
?	?	?	?
10	?	?	?
10	?	?	?
10	20	?	?
10	20	?	?
10	20	30	?
10	20	30	40
10	20	30	40

Midterm Pattern

- Section I (60%)
  - Output of C++ programs
  - Identifying and correcting errors in C++ Programs
- Section II (40 %)
  - Dry run of Pseudo code
  - Writing Pseudo code

Working with Characters and `string` Objects

- Using **`cin`** with the >> operator to input strings can cause problems:
  - Skips or stops on space, tab, end-of-line, end-of-file
  - Skips over leading white space;
  - Stops on trailing white space.
- To read any single char V (incl. whitespace)
  - **cin.get(V)**
- To skip input characters:
  - **cin.ignore( ); // one character.**
  - **cin.ignore(n); // n characters.**
- To work around this problem, you can use a C++ function named **`getline`**.

Working with Characters and **`string`** Objects

- Mixing `cin >>` and `cin.get()` in the same program can cause input errors that are hard to detect
- To skip over unneeded characters that are still in the keyboard buffer, use `cin.ignore()`:

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

String Input

- >> operator can NEVER read strings that contain WHITESPACE
  - **Skips** or **stops** on space, tab, end-of-line, end-of-file

- To read string S (which may contain whitespace)
  - string S;
  - getline(cin, S);

- How it works: reads all characters from cursor (5) to the end-of-line character (20), but does not store the eoln character.

  _ 1 2 3 _ T O M _ B R O W N _ 7 2 . 5 eol
  1 2 3 4 **5** 6 7 8 9 0 1 2 3 4 5 6 7 8 9 **20** ← position

  getline(cin, S); **S = " TOM BROWN 72.5"**

Using `getline` in Program 3-19

Program 3-19

```
1  // This program demonstrates using the getline function
2  // to read character data into a string object.
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9     string name;
10    string city;
11
12    cout << "Please enter your name: ";
13    getline(cin, name);
14    cout << "Enter the city you live in: ";
15    getline(cin, city);
16
17    cout << "Hello, " << name << endl;
18    cout << "You live in " << city << endl;
19    return 0;
20 }
```

Program Output with Example Input Shown in Bold

```
Please enter your name: Kate Smith [Enter]
Enter the city you live in: Raleigh [Enter]
Hello, Kate Smith
You live in Raleigh
```

Changing `cout` behaviour

- Use setf() and unsetf() to set following attributes
  - E.g. `cout.setf(ios::scientific);`

Flag	Meaning
`ios::showpoint`	display the decimal point
`ios::fixed`	fixed decimal notation
`ios::scientific`	scientific notation
`ios::right`	right justification
`ios::left`	left justification

`cout` Precision and justification

- With `#include <iomanip>` you can use
  - `setw(`*n*`)` and `setprecision(`*n*`)`
- E.g.

```
cin >> n;
cout << setprecision(4)
 << "Sqrt with 4 digits: " << sqrt(n)
 << endl << "Sqrt right justified: "
 << setw(10) << sqrt(n) << endl;
```

Trigonometric Functions

`sin(x)`	sine of x, where x is in radians
`cos(x)`	cosine of x, where x is in radians
`tan(x)`	tangent of x, where x is in radians
`asin(x)`	$\text{sine}^{-1}(x)$, returns angle in radians $[-\pi/2, \pi/2]$
`acos(x)`	$\text{cosine}^{-1}(x)$, returns angle in radians $[0,\pi]$
`atan(x)`	$\tan^{-1}(x)$, returns angle in radians $[-\pi/2, \pi/2]$
`atan2(y,x)`	$\tan^{-1}(y/x)$, returns angle in radians $[-\pi, \pi]$
`sinh(x)`	Hyperbolic sine of x
`cosh(x)`	Hyperbolic cosine of x
`tanh(x)`	Hyperbolic tan of x

Thank you