# Chapter # 3

Instructor: Sadullah Karimi, Msc in CSE

# Agenda

- **namespace and Using cin and cout in a Program**
- **Input (Read) Statement**
- **Variable Initialization**
- **Increment and Decrement Operators**
- **Commonly Used Escape Sequences**
- **Preprocessor Directives**
- **Creating a C++ Program**
- **Syntax**
- **Semantics**
- **Prompt Lines**
- **More on Assignment Statements**

# Input (Read) Statement

- Putting data into variables from the standard input device is accomplished via the use
- of cin and the operator >>. The syntax of cin together with >> is:

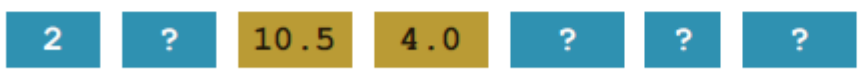
```
cin >> variable >> variable ...;
```

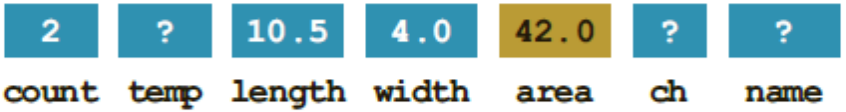
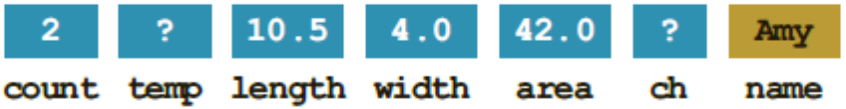
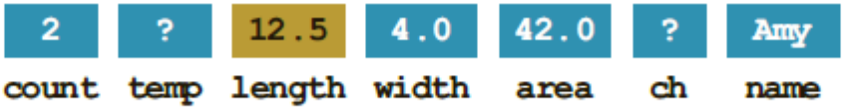
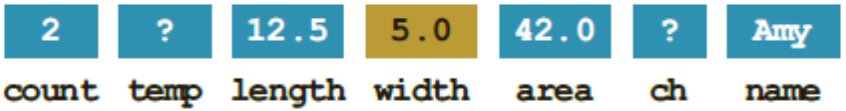
```
// This program illustrates how input statements work.
#include <iostream>
using namespace std;
int main()
{
int feet;
int inches;
cout << "Enter two integers separated by one or more spaces: ";
cin >> feet >> inches;
cout << endl;
cout << "Feet = " << feet << endl;
cout << "Inches = " << inches << endl;
return 0;
}
```

- This example further illustrates how assignment statements and input statements
- manipulate variables. Consider the following declarations:
- int count, temp;
- double length, width, area;
- char ch;
- string name;
- Also, suppose that the following statements execute in the order given.
- 1. count = 1;
- 2. count = count + 1;

- 3. cin >> length >> width;
- 4. area = length * width;
- 5. cin >> name;
- 6. length = length + 2;
- 7. width = 2 * length - 5 * width;
- 8. area = length * width;
- 9. cin >> ch;
- 10. temp = count + static_cast<int>(ch);

# Variable Initialization

| After St. | Values of the Variables/Statement | Explanation |
|---|---|---|
| 1 |  <br>`count = 1;` | Store `1` into `count`. |
| 2 |  <br>`count = count + 1;` | `count + 1 = 1 + 1 = 2`. Store `2` into `count`. This statement replaces the old value of `count` with this new value. |
| 3 |  <br>`cin >> length >> width;` | Read two numbers, which are `10.5` and `4.0`, and store the first number into `length`, and the second into `width`. |

| After St. | Values of the Variables/Statement | Explanation |
|---|---|---|
| 4 |  <br>`area = length * width;` | `length * width = 10.5 * 4.0 = 42.0`. Store `42.0` into `area`. |
| 5 |  <br>`cin >> name;` | Read the next input, **Amy**, from the keyboard and store it into `name`. |
| 6 |  <br>`length = length + 2;` | `length + 2 = 10.5 + 2 = 12.5`. Store `12.5` into `length`. This statement replaces the old value of `length` with this new value. |
| 7 |  <br>`width = 2 * length - 5 * width;` | `2 * length - 5 * width = 2 * 12.5 - 5 * 4.0 = 5.0`. Store `5.0` into `width`. This statement replaces the old value of `width` with this new value. |

| | | |
|---|---|---|
| 8 |  <br>`area = length * width;` | `length * width = 12.5 * 5.0 = 62.5`. Store `62.5` into `area`. This statement replaces the old value of `area` with this new value. |
| 9 |  <br>`cin >> ch;` | Read the next input, `A`, from the keyboard and store it into `ch`. |
| 10 |  <br>`temp = count + static_cast<int>(ch);` | `count + static_cast<int>ch) = 2 + static_cast<int>('A') = 2 + 65 = 67`. Store `67` into `temp`. |

# Increment and Decrement Operators

- Suppose count is an int variable. The statement:
- count = count + 1;
- Pre-increment: ++variable
- Post-increment: variable++
- The syntax of the decrement operator is:
- Pre-decrement: --variable
- Post-decrement: variable--

# Commonly Used Escape Sequences

TABLE 2-4 Commonly Used Escape Sequences

| | Escape Sequence | Description |
|---|---|---|
| `\n` | Newline | Cursor moves to the beginning of the next line |
| `\t` | Tab | Cursor moves to the next tab stop |
| `\b` | Backspace | Cursor moves one space to the left |
| `\r` | Return | Cursor moves to the beginning of the current line (not the next line) |
| `\\` | Backslash | Backslash is printed |
| `\'` | Single quotation | Single quotation mark is printed |
| `\"` | Double quotation | Double quotation mark is printed |

# namespace and Using cin and cout in a Program

- The using namespace std; statement should appear after the statement:
    - #include <iostream>
- You can then refer to cin and cout without using the prefix std::.
- To simplify the use of cin and cout, this book uses the second form.
- That is, to use cin and cout in a program,
- the programs will contain the following two statements:
    - #include <iostream>
    - using namespace std;

# Creating a C++ Program

```
int main()
{
     statement_1
          .
          .
          .
     statement_n

     return 0;
}
```

**EXAMPLE 2-28**

The following statements are examples of executable statements:

```
a = 4;                                  //assignment statement
cin >> b;                               //input statement
cout << a << " " << b << endl;          //output statement
```

# Syntax

- The syntax rules of a language tell what is legal and what is not legal. Errors in syntax are detected during compilation.
- For example, consider the following C++ statements:
- int x; //Line 1
- int y //Line 2
- double z; //Line 3
- y = w + x; //Line 4

# Continue Syntax

- When these statements are compiled, a compilation error will occur at Line 2 because the semicolon is missing after the declaration of the variable y.
- A second compilation error will occur at Line 4 because the identifier w is used but has not been declared.

# Use of Blanks

- In C++, you use one or more blanks to separate numbers when data is input.
- Blanks are also used to separate reserved words and identifiers from each other and from other symbols.
- Blanks must never appear within a reserved word or identifier.

# Use of Semicolons(;), Brackets({}), and Commas(,)

- The semicolon is also called a statement terminator.
- Note that curly braces, { and }, are not C++ statements in and of themselves, even though they often appear on a line with no other code.
- You might regard brackets as delimiters (boundary), because they enclose the body of a function and set it off from other partsof the program.
- Recall that commas (,) are used to separate items in a list.
- For example, you use commas when you declare more than one variable following a data type.

# Semantics

- The set of rules that gives meaning to a language is called semantics.
- For example, the order-of-precedence rules for arithmetic operators are semantic rules.
- For example, the following two lines of code are both syntactically correct expressions, but they have different meanings:
  - 2 + 3 * 5
  - And:
    - (2 + 3) * 5

# Prompt Lines

- Prompt lines are executable statements that inform the user what to do.
- For example, consider the following C++ statements, in which num is an int variable:
- cout << "Please enter an integer between 1 and 10 and " << "press the return key" << endl;
- cin >> num;
- When these two statements execute in the order given, first the output statement causes the following line of text to appear on the screen:
- **Please enter an integer between 1 and 10 and press the return key**

# Prompt Lines

- After seeing this line, users know that they must enter an integer and press the return key.
- If the program contained only the second statement, users would have no idea that they must enter an integer, and the computer would wait forever for the input.
- The preceding output statement is an example of a prompt line.
- In a program, whenever input is needed from users, you must include the necessary prompt lines.
- Furthermore, these prompt lines should include as much information as possible about what input is acceptable.
- For example, the preceding prompt line not only tells the user to input a number, but also informs the user that the number should be between 1 and 10.

# More on Assignment Statements

- x = x * y;
- Using the compound operator *=, this statement can be written as:
    - x *= y;
- variable = variable * (expression);
- as:
    - variable *= expression;

## EXAMPLE 2-31

This example shows several compound assignment statements that are equivalent to simple assignment statements.

| Simple Assignment Statement | Compound Assignment Statement |
| --- | --- |
| `i = i + 5;` | `i += 5;` |
| `counter = counter + 1;` | `counter += 1;` |
| `sum = sum + number;` | `sum += number;` |
| `amount = amount * (interest + 1);` | `amount *= interest + 1;` |
| `x = x / (y + 5);` | `x /= y + 5;` |

- Any question are appreciating