C++/C Basics

Prof. Seokin Hong

Agenda

- Variables and Assignments
- Input and Output
- Data Types and Expressions
- Flow of Control
- Designing Loops

Variables and Assignments

Variable

- **Variables are names for memory locations**
- **Choosing variable names**
  - Use meaningful names that represent data
  - The first character must be
    - a letter (alphabet)
    - the underscore character (_)
  - The remaining characters must be
    - letters (alphabet)
    - numbers
    - underscore character (_)

```
#include <iostream>
using namespace std;
int main( )
{
int     number_of_bars;
double one_weight, total_weight;

  cout << "Enter the number of candy bars in a package\n";
  cout << "and the weight in ounces of one candy bar.\n";
  cout << "Then press return.\n";
  cin >> number_of_bars;
```

Declaring Variables

- **Before use, variables must be declared**
  - Tells the compiler the data type

    Examples: int number_of_bars;
    double one_weight, total_weight;

  - int is an abbreviation for integer.
    - could store 3, 102, 3211, -456, etc.
  - double represents numbers with a fractional component
    - could store 1.34, 4.0, -345.6, etc.

```
#include <iostream>
using namespace std;
int main( )
{
int      number_of_bars;
double one_weight, total_weight;

  cout << "Enter the number of candy bars in a package\n";
  cout << "and the weight in ounces of one candy bar.\n";
  cout << "Then press return.\n";
  cin >> number_of_bars;
```

Declaring Variables (Cont'd)

- **Two locations for variable declarations**

Immediately prior to use

```
int main()
 {
     …
     int sum;
     sum = score1 + score 2;
     …
     return 0;
 }
```

At the beginning

```
int main()
{
     int sum;
        …
        sum = score1 + score2;

         …
         return 0;
      }
```

Assignment Statements

- **An assignment statement changes the value of a variable**
  - totalWeight = oneWeight + numberOfBars;

  - On the right of the assignment operator can be
    - **Constants** → age = 21;
    - **Variables** → myCost = yourCost;
    - **Expressions** → circumference = diameter * 3.14159;

Initializing Variables

- **Variables are initialized in assignment statements**

```
double mpg;        // declare the variable
mpg = 26.3;        // initialize the variable
```

- **Declaration and initialization can be combined using two methods**
  - Method 1

    ```
    double mpg = 26.3, area = 0.0 , volume;
    ```

  - Method 2 (C++ style)

    ```
    double mpg(26.3),  area(0.0), volume;
    ```

Input and Output

Output using cout

- cout is an output stream sending data to the monitor
  - cout is an object of ostream class. Defined in iostream header file
- The insertion operator "<<" inserts data into cout
- Example:

```
cout << numberOfBars << " candy bars\n";

cout << numberOfBars;
cout << " candy bars\n";
```

- Arithmetic is performed in the cout statement

```
cout << "Total cost is $" << (price + tax);
```

How does operator << handle multiple data type?

Include Directives

- **“Include” directives add library files to our programs**
  - To make the definitions of the cin and cout available to the program:

```
#include <iostream>
```

- **“Using” directives include a collection of defined names**
  - To make the names cin and cout available to our program:

```
using namespace std;
```

Formatting Real Numbers

- **Real numbers (type double) produce a variety of outputs**

```
double price = 78.51;
cout << "The price is $" << price << endl;
```

  - The output could be any of these:

```
The price is $78.51
The price is $78.510000
The price is $7.851000e01
```

```
Example:   cout.setf(ios::fixed);
           cout.precision(2);
           cout        << "The price is "
                       << price << endl;
```

- **Fixed point notation VS scientific notation**
  - cout.setf(ios::fixed), cout.setf(ios::scientific)
- **To specify that two decimal places will always be shown**
  - precision(2)

https://www.cplusplus.com/reference/ios/ios_base/setf/?kw=setf

Input Using cin

- cin is an input stream bringing data from the keyboard
- The extraction operator (>>) get data from the input stream
- Example:

```
cout << "Enter the number of bars in a package\n";
cout << " and the weight in ounces of one bar.\n";
cin >> numberOfBars;
cin >> oneWeight;
```

- Multiple data items are separated by spaces
- Data is not read until the enter key is pressed

Example:

```
    cin >> v1 >> v2 >> v3;
* User might type
            34  45  12   <enter key>
```

Data Types and Expressions

Integer types

- **long or long int (often 4 bytes)**
  - Declare very large integers

```
long  bigTotal;
long int bigTotal;
```

- **short or short int (often 2 bytes)**
  - Declare smaller integers

```
short smallTotal;
short int smallTtotal;
```

Floating point types

- **double (often 8 bytes), long double (often 16 bytes)**
  - Declares very large floating point numbers

```
double bigNumber;
```

- **float (often 4 bytes)**
  - Declares smaller floating point numbers

```
float notSoBigNumber;
```

char

- **char**
    - Can be any single character
- **To declare a variable of type char:**

```
char letter;
```

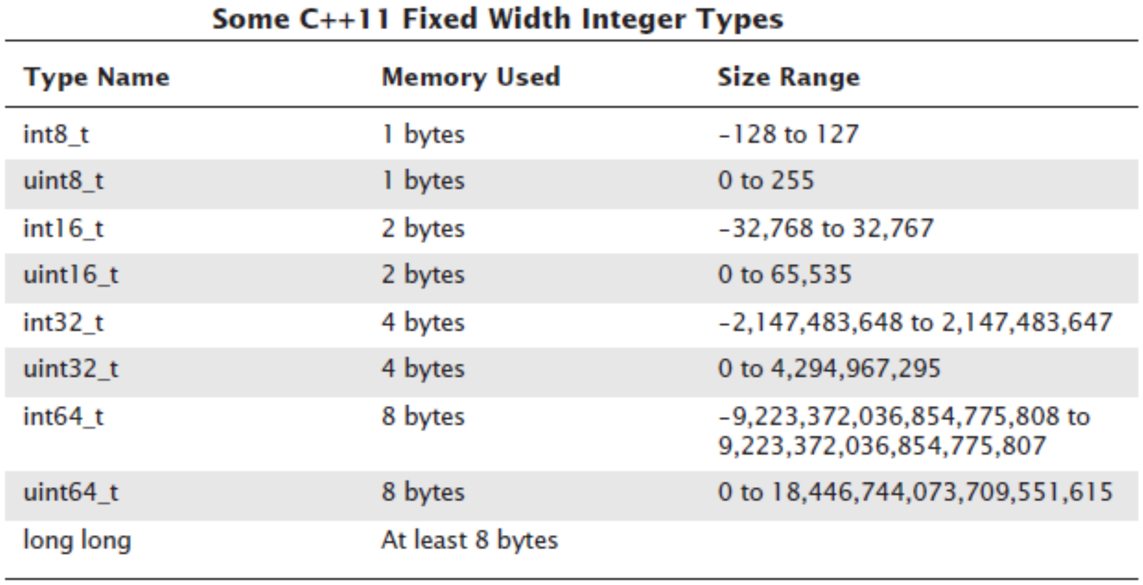
- **Character constants are enclosed in single quotes**

```
char letter = 'a’;
```

- **Strings of characters: enclosed in double quotes**
    - "a" is a string of characters containing one character
    - 'a' is a value of type character

C++11 Types

- C++11 introduced new integer types that specify **exactly the size** and **whether or not the data type is signed or unsigned**



Some C++11 Fixed Width Integer Types

Type Name	Memory Used	Size Range
int8_t	1 bytes	-128 to 127
uint8_t	1 bytes	0 to 255
int16_t	2 bytes	-32,768 to 32,767
uint16_t	2 bytes	0 to 65,535
int32_t	4 bytes	-2,147,483,648 to 2,147,483,647
uint32_t	4 bytes	0 to 4,294,967,295
int64_t	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
uint64_t	8 bytes	0 to 18,446,744,073,709,551,615
long long	At least 8 bytes	

Compile for c++11

```
$ g++ -std=c++11 test.cc
```

C++11 Types (Cont.)

- **auto**
  - deduces the type of the variable based on the expression on the right hand side of the assignment
  - Example
    - auto x = 78.51;   // x becomes a double
    - auto x = 78;   //x becomes a int

- **bool**
  - **true or false**
    - **true: 1**
    - **false: 0**
  - Usually, used in branching and looping statements

Enumeration Types

- **An enumeration type is a type with values defined by a list of constants**

- **Example:**

  -
    ```
    enum MonthLength{JAN_LENGTH = 31,
                     FEB_LENGTH = 28,
                     MAR_LENGTH = 31,
                             …
                      DEC_LENGTH = 31};
    ```

```
#include<stdio.h>

enum MonthLength{JAN_LENGTH = 31,
                             FEB_LENGTH = 28,
                               MAR_LENGTH = 31,
                                          …
                                DEC_LENGTH = 31};

int main()
{
    enum MonthLength month_length;
    month_length = JAN_LENGTH;
    printf("%d", month_length);
    return 0;
}
```

Enumeration Types (Cont.)

- **Default enum Values**
  - If numeric values are not specified, identifiers are assigned consecutive values starting with 0

```
enum Direction { NORTH = 0, SOUTH = 1, EAST = 2, WEST = 3};
```

is equivalent to

```
enum Direction {NORTH, SOUTH, EAST, WEST};
```

Enumeration Types (Cont.)

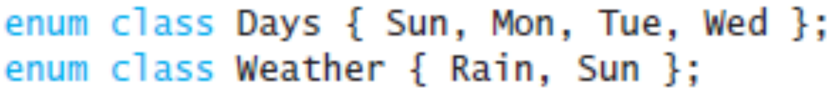
- **Unless specified, the value assigned an enumeration constant is 1 more than the previous constant**

```
enum MyEnum{ONE = 17, TWO, THREE, FOUR = -3, FIVE};
```

results in these values

ONE = 17, TWO = 18, THREE = 19,
FOUR = -3, FIVE = -2

Enumeration Types (Cont.)



```
enum class Days { Sun, Mon, Tue, Wed };
enum class Weather { Rain, Sun };
```

- **C++11 introduced enum class**
- **some problems of conventional enums**
  - enums are global so you can’t have the same enum value twice
  - may not want an enum to act like an int
- **Define an enum class as follows:**

```
enum class Colors {RED, BLUE, GREEN, YELLOW,}
enum class RainbowColors {RED, ORANGE}
```

- **To use the strong enums:**

```
Color d = Colors::RED;

RainbowColor w = RainbowColors::RED;
```

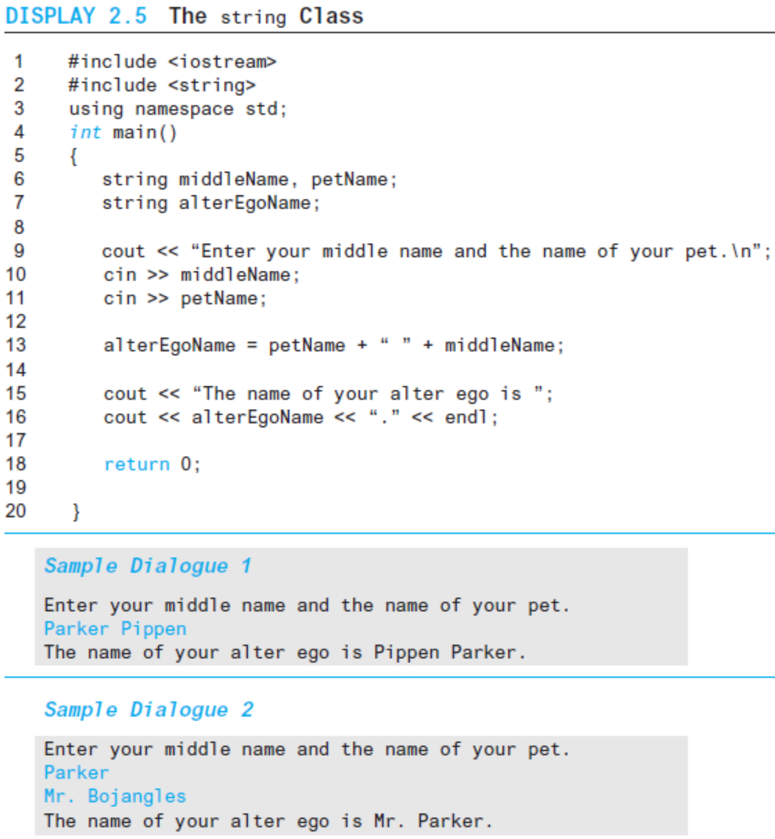
string

- string is a class, different from the primitive data types discussed so far
  - Use double quotes around the text to store into the string variable
  - Requires the following be added to the top of your program:

    #include <string>

- To declare a variable of type string:

  string name = "Apu Nahasapeemapetilon";



DISPLAY 2.5 The `string` Class

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string middleName, petName;
    string alterEgoName;

    cout << “Enter your middle name and the name of your pet.\n”;
    cin >> middleName;
    cin >> petName;

    alterEgoName = petName + “ ” + middleName;

    cout << “The name of your alter ego is ”;
    cout << alterEgoName << “.” << endl;

    return 0;

}
```

Sample Dialogue 1

```
Enter your middle name and the name of your pet.
Parker Pippen
The name of your alter ego is Pippen Parker.
```

Sample Dialogue 2

```
Enter your middle name and the name of your pet.
Parker
Mr. Bojangles
The name of your alter ego is Mr. Parker.
```

Type Compatibilities

- **Variables of type double should not be assigned to variables of type int**

```
int intVariable;
double doubleVariable;
doubleVariable = 2.00;
intVariable = doubleVariable;     //intVariable contains 2, not 2.00
```

- **Integer values can normally be stored in variables of type double**

```
double doubleVariable;
doubleVariable = 2;
```

- **It is possible to store char values in integer variables**

```
int value = 'A';
```

- **It is possible to store int values in char variables**

```
char letter = 65;
```

Arithmetic

- **Arithmetic is performed with operators**
  - + for addition
  - - for subtraction
  - * for multiplication
  - / for division

- **Example:**

```
totalWeight  =  oneWeight * numberOfBars;
```

Division of Doubles

- **Division with at least one operator of type double produces the expected results.**

```
double divisor, dividend, quotient;
divisor = 3;
dividend = 5;
quotient = dividend / divisor;
```

  - quotient = 1.6666…
  - Result is the same if either dividend or divisor is of type int

Division of Integers

- **Be careful with the division operator!**
  - int / int produces an integer result

```
int dividend, divisor, quotient;
 dividend = 5;
 divisor = 3;
quotient = dividend / divisor;
```

  - The value of quotient is 1, not 1.666… → the fractional part is discarded!

Integer Remainders

- **% operator gives the remainder from integer division**

```
int dividend, divisor, remainder;
dividend = 5;
divisor = 3;
remainder = dividend % divisor;
```

The value of remainder is 2

Operator Shorthand

- **All arithmetic operators can be used this way**
  - += count = count + 2; becomes
    count += 2;
  - *= bonus = bonus * 2; becomes
    bonus *= 2;
  - /= time = time / rushFactor; becomes
    time /= rushFactor;
  - %= remainder = remainder % (cnt1+ cnt2); becomes
    remainder %= (cnt1 + cnt2);

Flow of Control

if-else Flow Control

- **Syntax**

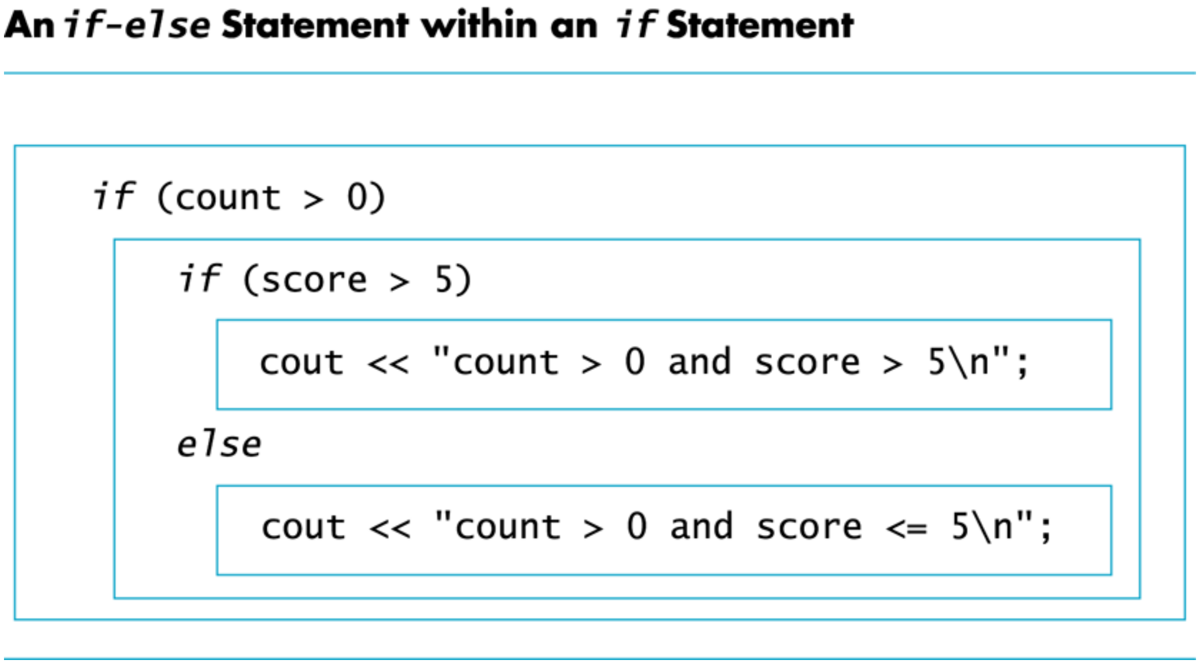
```
if (boolean expression){
        true statements
}
else{
        false statements
 }
```

When the boolean expression is true

When the boolean expression is false

Nested Statements

- A statement that is a subpart of another statement is a nested statement

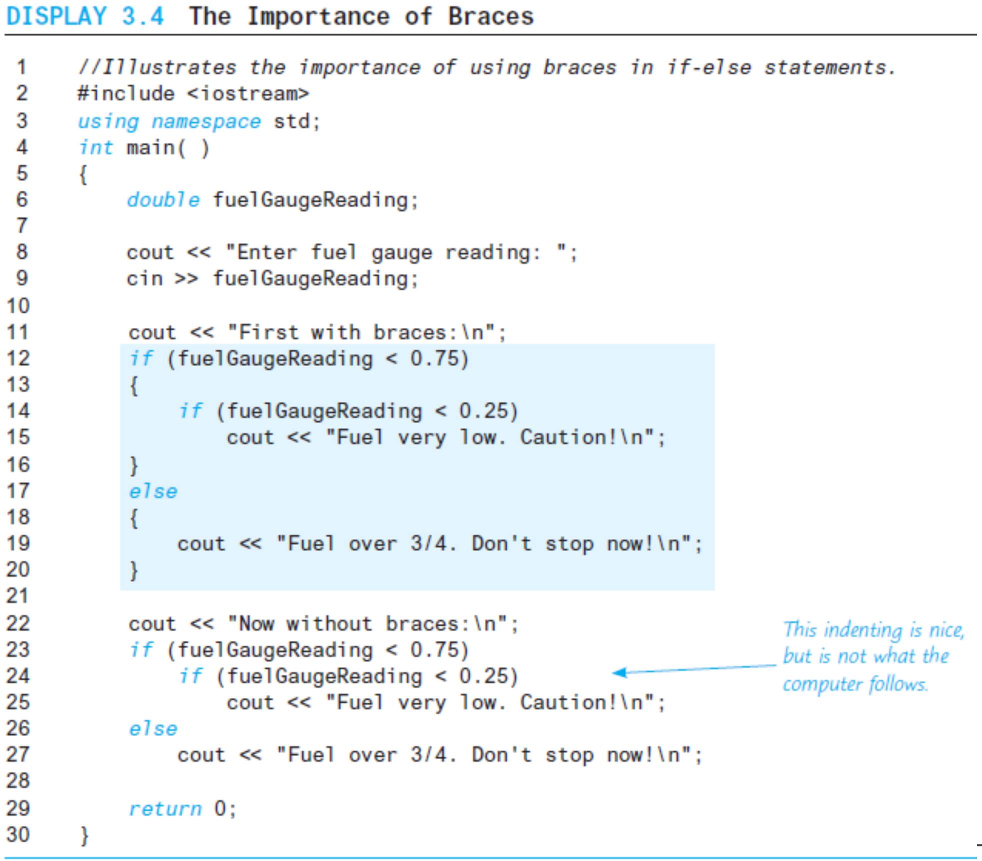


An *if-else* Statement within an *if* Statement

```
if (count > 0)
    if (score > 5)
        cout << "count > 0 and score > 5\n";
    else
        cout << "count > 0 and score <= 5\n";
```

Braces and Nested Statements

- **Braces in nested statements are like parenthesis in arithmetic expressions**
  - Braces tell the compiler how to group things



DISPLAY 3.4 The Importance of Braces

```
//Illustrates the importance of using braces in if-else statements.
#include <iostream>
using namespace std;
int main( )
{
    double fuelGaugeReading;

    cout << "Enter fuel gauge reading: ";
    cin >> fuelGaugeReading;

    cout << "First with braces:\n";
    if (fuelGaugeReading < 0.75)
    {
        if (fuelGaugeReading < 0.25)
            cout << "Fuel very low. Caution!\n";
    }
    else
    {
        cout << "Fuel over 3/4. Don't stop now!\n";
    }

    cout << "Now without braces:\n";
    if (fuelGaugeReading < 0.75)
        if (fuelGaugeReading < 0.25)
            cout << "Fuel very low. Caution!\n";
    else
        cout << "Fuel over 3/4. Don't stop now!\n";

    return 0;
}
```

This indenting is nice, but is not what the computer follows.

Multi-way if-else-statements

- An if-else-statement is a two-way branch
- Three or four (or more) way branches can be designed using nested if-else-statements

Two-way branch

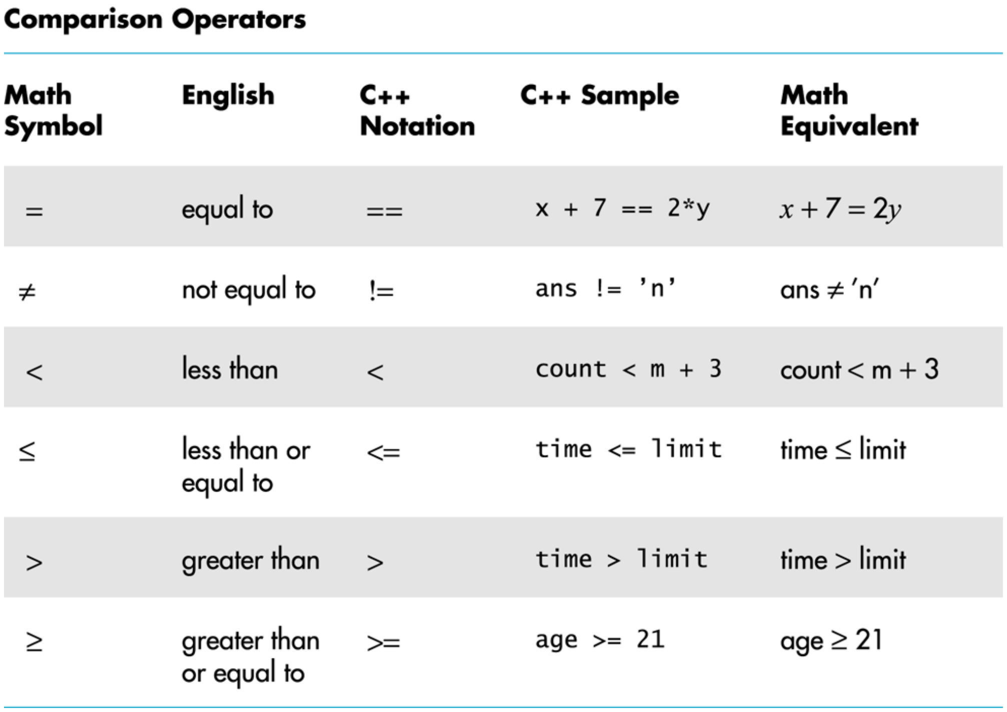
```
if (guess> number)
   cout << "Too high.";
else
    if (guess < number)
        cout << "Too low.");
    else
        if (guess == number)
            cout << "Correct!";
```

Multi-way branch

```
if (guess> number)
      cout << "Too high.";
else if (guess < number)
      cout << "Too low.");
else if (guess == number)
      cout << "Correct!";
```

Boolean Expressions

- Boolean expressions are expressions that are either true or false



Comparison Operators

Math Symbol	English	C++ Notation	C++ Sample	Math Equivalent
=	equal to	==	`x + 7 == 2*y`	$x + 7 = 2y$
≠	not equal to	!=	`ans != 'n'`	ans ≠ 'n'
<	less than	<	`count < m + 3`	count < m + 3
≤	less than or equal to	<=	`time <= limit`	time ≤ limit
>	greater than	>	`time > limit`	time > limit
≥	greater than or equal to	>=	`age >= 21`	age ≥ 21

Boolean Expressions (Cont.)

- **AND**
  - Boolean expressions can be combined into more complex expressions with
  - **Syntax**: (Comparison_1) && (Comparison_2)
  - **Example**: if ((2 < x) && (x < 7))
    - True only if x is between 2 and 7

- **OR**
  - True if either or both expressions are true
  - **Syntax**: (Comparison_1) | | (Comparison_2)

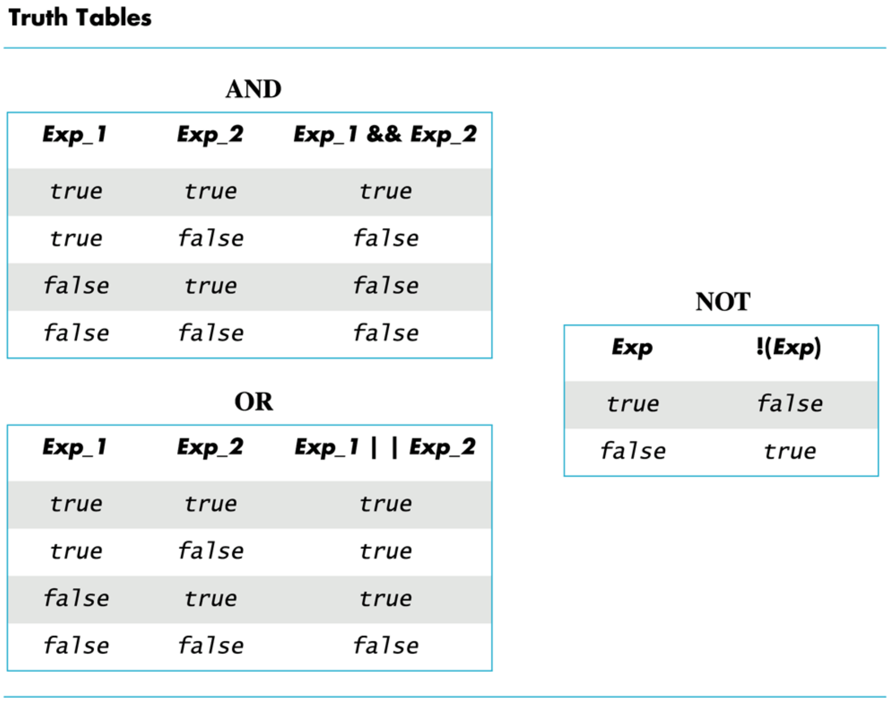
  - **Example**: if ((x = = 1) | | (x = = y))

Boolean Expressions (Cont.)

- **NOT**
  - ! -- negates any boolean expression
  - **Example**
    - !(x < y)
      - True if x is NOT less than y
    - !(x = = y)
      - True if x is NOT equal to y

Evaluating Boolean Expressions

- Boolean expressions are evaluated using values from the Truth Tables in



Truth Tables

AND

Exp_1	Exp_2	Exp_1 && Exp_2
true	true	true
true	false	false
false	true	false
false	false	false

OR

Exp_1	Exp_2	Exp_1 | | Exp_2
true	true	true
true	false	true
false	true	true
false	false	false

NOT

Exp	!(Exp)
true	false
false	true

For example, if y is 8, the expression

!((y < 3) | | (y > 7))

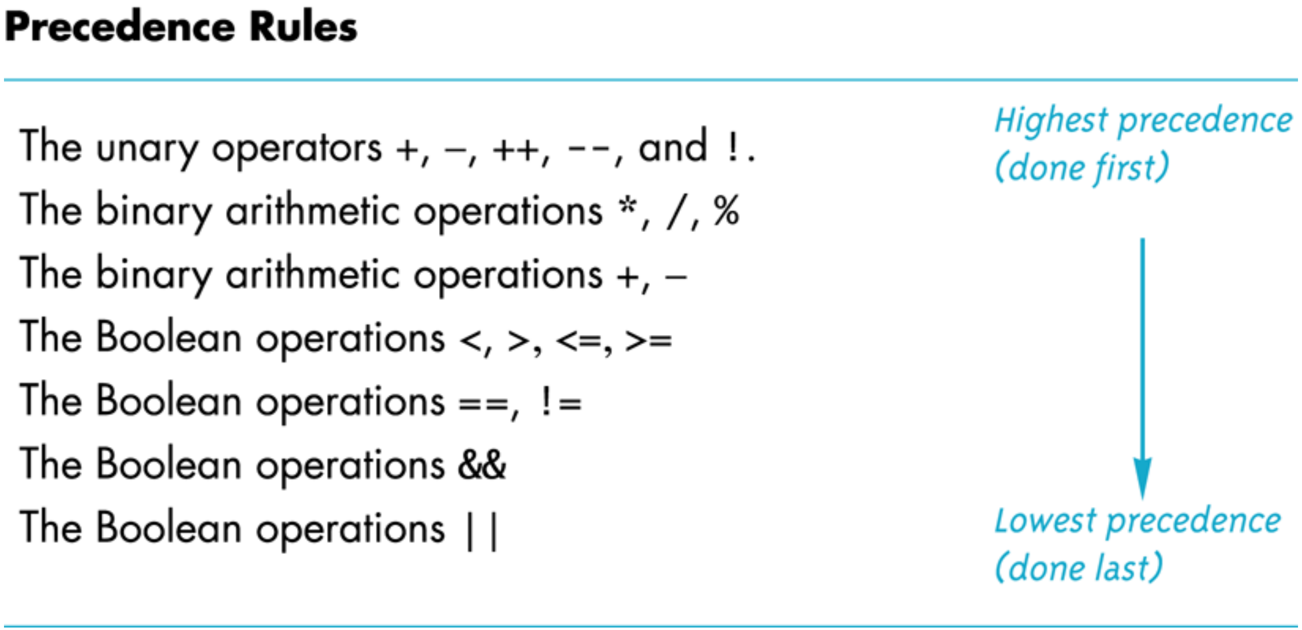
is evaluated in the following sequence

! (false | | true)

! (true)

false

Precedence Rules



Precedence Rules

The unary operators +, –, ++, – –, and ! .

The binary arithmetic operations *, /, %

The binary arithmetic operations +, –

The Boolean operations <, >, <=, >=

The Boolean operations ==, !=

The Boolean operations &&

The Boolean operations | |

Highest precedence (done first)

Lowest precedence (done last)

Precedence Rules (Cont.)

- The expression

  (x+1) > 2 | | (x + 1) < -3

  is equivalent to

  ((x + 1) > 2) | | ((x + 1) < -3)

  Because > and < have higher precedence than | |

  and is also equivalent to

  x + 1 > 2 | | x + 1 < - 3

Short-Circuit Evaluation

- **C++ uses short-circuit evaluation**
  - If the value of the leftmost sub-expression determines the final value of the expression, the rest of the expression is not evaluated
  - **Example**:
    - if x is negative, the value of the expression

      (x >= 0) && (y > 1)

      can be determined by evaluating only (x >= 0)

switch-statement

```
switch (controlling expression)
{
     case Constant_1:
                    statement_Sequence_1
                    break;
      case Constant_2:
                    Statement_Sequence_2
                    break;
                 . . .
      case Constant_n:
                    Statement_Sequence_n
                    break;
      default:
                    Default_Statement_Sequence
}
```

must return one of these types

1.bool value
2.enum constant
3.integer type
4.character

The value returned is compared to the constant values after each "case"

switch-statement (cont.)

- **The break statement ends the switch-statement**
  - Omitting the break statement will cause the code for the next case to be executed
  - Omitting a break statement allows the use of multiple case labels for a section of code

    -
      ```
      case 'A':
      case 'a':
                   cout << "Excellent.";
                  break;
      ```
    - Runs the same code for either 'A' or 'a'

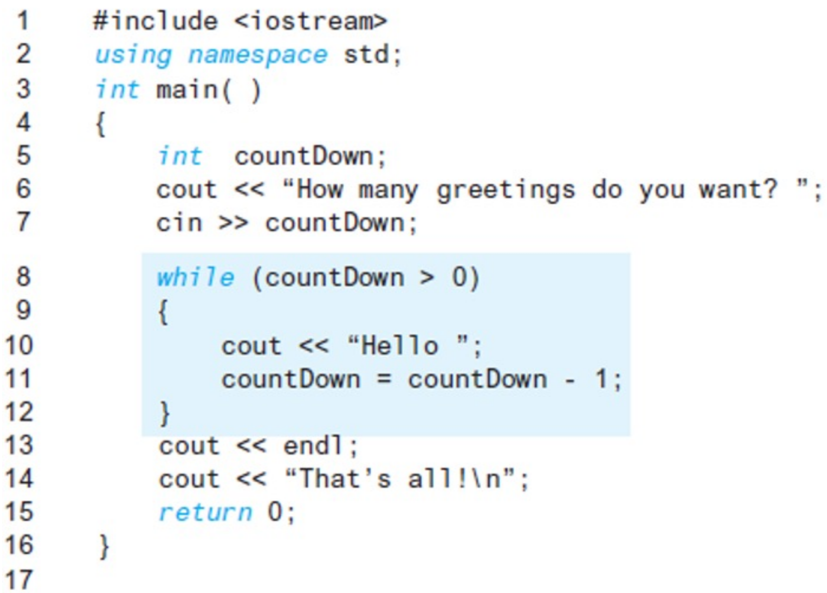
switch-statement (cont.)

- **Default label**
  - If no case label has a constant that matches the controlling expression, the statements following the default label are executed
  - If there is no default label, nothing happens when the switch statement is executed
  - It is a good idea to include a default section!!

while-loop

- When an action must be repeated, a loop is used
- Example:

```
while (countDown > 0)
 {
        cout << "Hello ";
        countDown  -= 1;
 }
```



```
#include <iostream>
using namespace std;
int main( )
{
    int  countDown;
    cout << “How many greetings do you want? ”;
    cin >> countDown;

    while (countDown > 0)
    {
        cout << “Hello ”;
        countDown = countDown - 1;
    }
    cout << endl;
    cout << “That’s all!\n”;
    return 0;
}
```

do-while loop

- **A variation of the while loop.**
- **A do-while loop is always executed at least once**
  - The body of the loop is first executed
  - The boolean expression is checked after the body has been executed

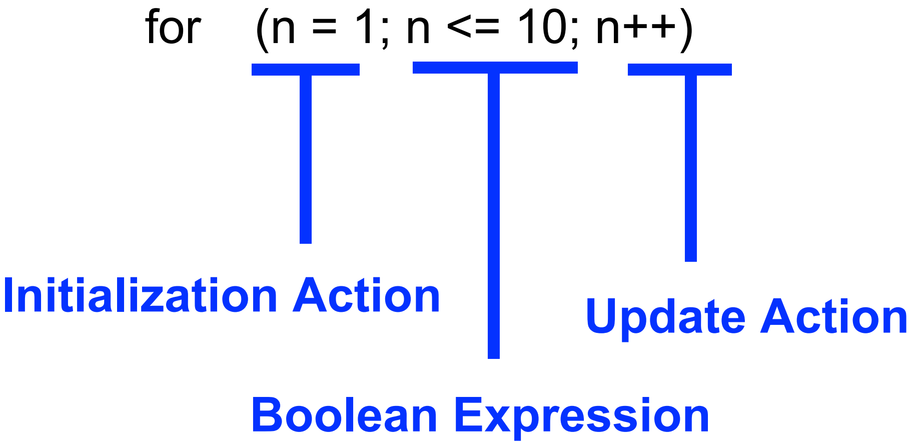
- **Syntax:**

```
do
 {
      statements to repeat

  }  while (boolean_expression);
```

For-loop

- **Syntax:**



```
for    (n = 1; n <= 10; n++)
```

Initialization Action

Boolean Expression

Update Action

```
sum = 0;
for (n = 1; n <= 10; n++)  //add the numbers 1 - 10
  sum = sum + n;


sum = 0;
n = 1;
while(n <= 10)  // add the numbers 1 - 10
 {
    sum = sum + n;
    n++;
  }
```

For-loop (Cont.)

- **Initialization and update actions of for-loops often contain more complex expressions**
  - Here are some samples

```
for (n = 1; n < = 10; n = n + 2)

for(n = 0 ; n > -100 ; n = n -7)

for(double x = pow(y,3.0);  x > 2.0;  x = sqrt(x) )
```

number++ vs ++number

- **number++** vs **++number**
  - (number++) returns the **current** value of number, then increments number
  - (++number) increments number first and returns the **new** value of number

  - **Example**

```
int number = 2;
int valueProduced = 2 * (number++);
cout << valueProduced << " " << number;
```

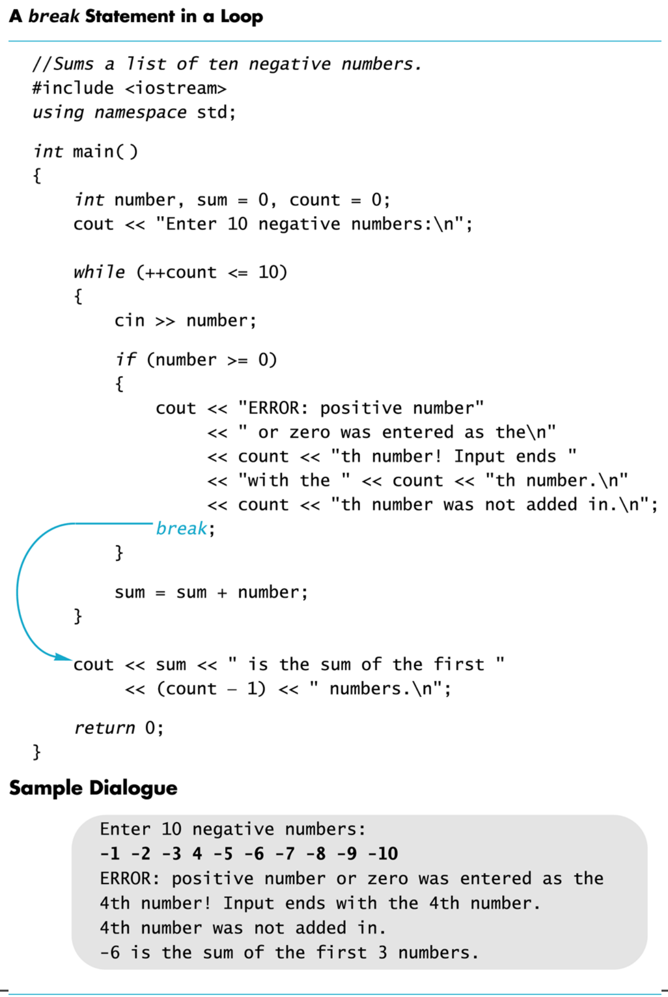
displays 4 3

```
int number = 2;
int valueProduced = 2* (++number);
cout << valueProduced << " " number;
```

displays 6 3

The break-Statement

- **The break-statement can be used to exit a loop before normal termination**
  - Be careful with nested loops!
    - Using break only exits the loop in which the break-statement occurs



A *break* Statement in a Loop

```
//Sums a list of ten negative numbers.
#include <iostream>
using namespace std;

int main( )
{
    int number, sum = 0, count = 0;
    cout << "Enter 10 negative numbers:\n";

    while (++count <= 10)
    {
        cin >> number;

        if (number >= 0)
        {
            cout << "ERROR: positive number"
                 << " or zero was entered as the\n"
                 << count << "th number! Input ends "
                 << "with the " << count << "th number.\n"
                 << count << "th number was not added in.\n";
            break;
        }

        sum = sum + number;
    }

    cout << sum << " is the sum of the first "
         << (count - 1) << " numbers.\n";

    return 0;
}
```

Sample Dialogue

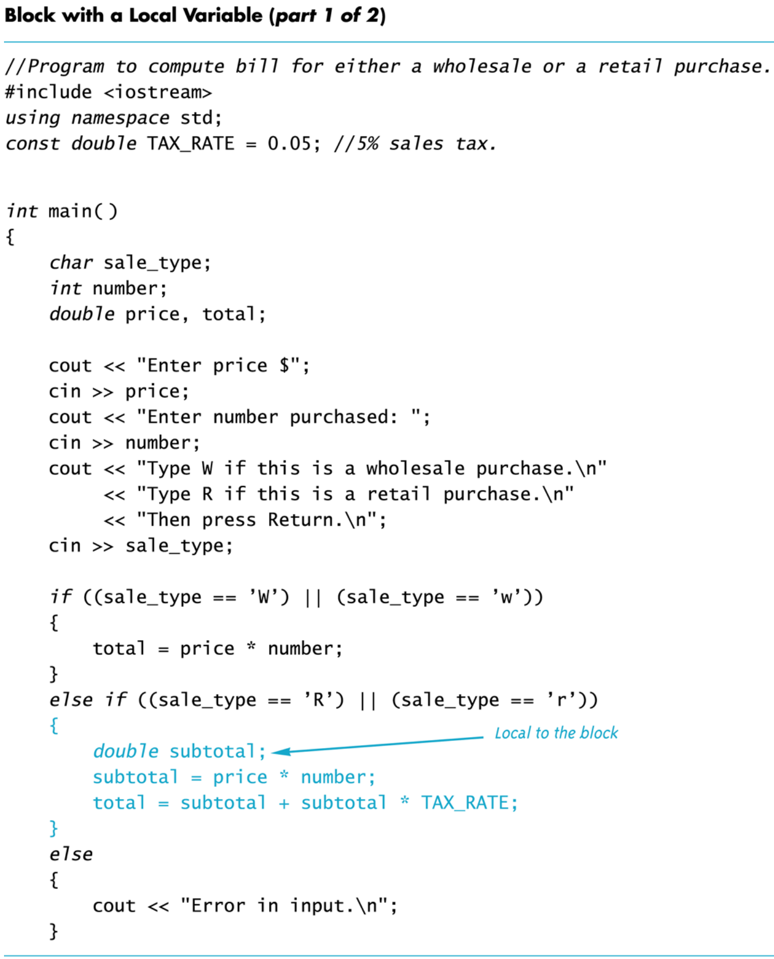
```
Enter 10 negative numbers:
-1 -2 -3 4 -5 -6 -7 -8 -9 -10
ERROR: positive number or zero was entered as the
4th number! Input ends with the 4th number.
4th number was not added in.
-6 is the sum of the first 3 numbers.
```

Scope Rule for Nested Blocks

- **Block**
  - A block is a section of code enclosed by braces

- **A variable declared outside of block can be accessed inside of block**

- **A variable declared inside of block cannot be accessed outside of block**



Block with a Local Variable (*part 1 of 2*)

```cpp
//Program to compute bill for either a wholesale or a retail purchase.
#include <iostream>
using namespace std;
const double TAX_RATE = 0.05; //5% sales tax.


int main( )
{
    char sale_type;
    int number;
    double price, total;

    cout << "Enter price $";
    cin >> price;
    cout << "Enter number purchased: ";
    cin >> number;
    cout << "Type W if this is a wholesale purchase.\n"
         << "Type R if this is a retail purchase.\n"
         << "Then press Return.\n";
    cin >> sale_type;

    if ((sale_type == 'W') || (sale_type == 'w'))
    {
        total = price * number;
    }
    else if ((sale_type == 'R') || (sale_type == 'r'))
    {
        double subtotal;
        subtotal = price * number;
        total = subtotal + subtotal * TAX_RATE;
    }
    else
    {
        cout << "Error in input.\n";
    }
```

Program Style - Comments

- **// is the symbol for a single line comment**
  - Comments are explanatory notes for the programmer
  - All text on the line following // is ignored by the compiler
  - Example:

    ```
    //calculate regular wages
    grossPay = rate * hours;
    ```

- **/*   and */ enclose multiple line comments**
  - Example:

    ```
    /*  This is a comment that spans
         multiple lines without a
         comment symbol on the middle line
    */
    ```

Program Style - Constants

- **Number constants used throughout a program are difficult to find and change when needed**
- **Constants**
  - Allow us to name number constants so they have meaning
  - Allow us to change all occurrences simply by changing the value of the constant

- **const is the keyword to declare a constant**
  - **Example**:

    ```
    const int WINDOW_COUNT = 10;
    ```

    - declares a constant named WINDOW_COUNT
    - Its value cannot be changed by the program like a variable
    - It is common to name constants with all capitals

Designing Loops

Designing Loops

- **Designing a loop involves designing**
  - The body of the loop
  - The initializing statements
  - The conditions for ending the loop

Sums and Products

- **A common task is reading a list of numbers and computing the sum**
  - Pseudocode for this task might be:

```
sum = 0;
repeat the following this_many times
        cin >> next;
        sum = sum + next;
end of loop
```

for-loop for a sum

- The pseudocode from the previous slide is implemented as

```
int sum = 0;
for(int count=1; count <= this_many; count++)
{
      cin >> next;
      sum = sum + next;
 }
```

for-loop For a Product

- **Forming a product is very similar to the sum example**

```
int product = 1;
for(int count=1; count <= this_many; count++)
{
        cin >> next;
        product = product * next;
 }
```

  - product must be initialized prior to the loop body
  - Notice that product is initialized to 1, not 0!

Ending a Loop

- **There are four common methods to terminate an input loop**
  - List headed by size
    - When we can determine the size of the list beforehand
  - Ask before iterating
    - Ask if the user wants to continue before each iteration
  - List ended with a sentinel value
    - Using a particular value to signal the end of the list
  - Running out of input
    - Using the eof function to indicate the end of a file

List Headed By Size

- **The for-loops we have seen provide a natural implementation of the list headed by size method of ending a loop**
  - Example:

```
int items;
cout << "How many items in the list?";
cin >> items;
for(int count  = 1; count <= items; count++)
{
        int number;
        cout << "Enter number " << count;
        cin >> number;
        cout << endl;          // statements to process the number
 }
```

Ask Before Iterating

- A while loop is used here to implement the ask before iterating method to end a loop

```
sum = 0;
cout << "Are there numbers in the list (Y/N)?";
char ans;
cin >> ans;

while (( ans = 'Y')  || (ans = 'y'))
{
      //statements to read and process the number
      cout << "Are there more numbers(Y/N)? ";
      cin >> ans;
 }
```

List Ended With a Sentinel Value

- A while loop is typically used to end a loop using the list ended with a sentinel value method

```
cout << "Enter a list of nonnegative integers.\n"
       << "Place a negative integer after the list.\n";
sum = 0;
cin >> number;
while (number > 0)
{
              //statements to process the number
              cin >> number;
}
```

Running Out of Input

- The while loop is typically used to implement the running out of input method of ending a loop

```
ifstream infile;
infile.open("data.dat");
while (! infile.eof( ) )
{
        // read and process items from the file
        // File I/O covered in Chapter 6
}
infile.close( );
```

Copyright Notice

- The contents of this slide deck are taken from the textbook (Problem Solving with C++, Walter Savitch).
- See your textbook for more details.
- Redistribution of this slide deck is not permitted.

NEXT ?

Functions