Chapter 2

C++ Basics

Overview

2.1

Variables and Assignments

Variables and Assignments

- Variables are like small blackboards
  - We can write a number on them
  - We can change the number
  - We can erase the number
- C++ variables are names for memory locations
  - We can write a value in them
  - We can change the value stored there
  - We cannot erase the memory location
    - Some value is always there

A C++ Program (*part 1 of 2*)

```
#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;
    cin >> one_weight;

    total_weight = one_weight * number_of_bars;

    cout << number_of_bars << " candy bars\n";
    cout << one_weight << " ounces each\n";
    cout << "Total weight is " << total_weight << " ounces.\n";

    cout << "Try another brand.\n";
    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;
    cin >> one_weight;

    total_weight = one_weight * number_of_bars;

    cout << number_of_bars << " candy bars\n";
    cout << one_weight << " ounces each\n";
    cout << "Total weight is " << total_weight << " ounces.\n";

    cout << "Perhaps an apple would be healthier.\n";

    return 0;
}
```

A C++ Program (*part 2 of 2*)

Sample Dialogue

```
Enter the number of candy bars in a package
and the weight in ounces of one candy bar.
Then press return.
11 2.1
11 candy bars
2.1 ounces each
Total weight is 23.1 ounces.
Try another brand.
Enter the number of candy bars in a package
and the weight in ounces of one candy bar.
Then press return.
12 1.8
12 candy bars
1.8 ounces each
Total weight is 21.6 ounces.
Perhaps an apple would be healthier.
```

Identifiers

- Variables names are called identifiers
- Choosing variable names
  - Use meaningful names that represent data to be stored
  - First character must be
    - a letter
    - the underscore character
  - Remaining characters must be
    - letters
    - numbers
    - underscore character

Keywords

- Keywords (also called reserved words)
  - Are used by the C++ language
  - Must be used as they are defined in the programming language
  - Cannot be used as identifiers

Declaring Variables (Part 1)

- Before use, variables must be declared
  - Tells the compiler the type of data to store

    Examples:
    ```
    int       number_of_bars;
    double one_weight,  total_weight;
    ```
  - int is an abbreviation for integer.
    - could store 3, 102, 3211, -456, etc.
    - number_of_bars   is of type integer
  - double represents numbers with a fractional component
    - could store 1.34, 4.0, -345.6, etc.
    - one_weight and total_weight are both of type double

Declaring Variables (Part 2)

- Immediately prior to use

Two locations for variable declarations

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

- At the beginning

```
int main()
{
     int sum;
    …
    sum = score1 +
             score2;
     …
    return 0;
 }
```

Declaring Variables (Part 3)

- Declaration syntax:
  - Type_name Variable_1 , Variable_2, . . . ;

- Declaration Examples:
  - double average, m_score, total_score;
  - double moon_distance;
  - int age, num_students;
  - int cars_waiting;

Assignment Statements

- An assignment statement changes the value of a variable
  - total_weight = one_weight + number_of_bars;
    - total_weight is set to the sum one_weight + number_of_bars
  - Assignment statements end with a semi-colon
  - The single variable to be changed is always on the left of the assignment operator ‘=‘
  - On the right of the assignment operator can be
    - Constants --   age = 21;
    - Variables --   my_cost = your_cost;
    - Expressions -- circumference = diameter * 3.14159;

Assignment Statements and Algebra

- The ‘=‘ operator in C++ is not an equal sign
  - The following statement cannot be true in algebra
    - number_of_bars = number_of_bars + 3;
  - In C++ it means the new value of number_of_bars
    is the previous value of number_of_bars plus 3

Initializing Variables

- Declaring a variable does not give it a value
  - Giving a variable its first value is initializing the variable
- 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

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

Section 2.1 Conclusion

- Can you
  - Declare and initialize two integers variables to zero? The variables are named feet and inches.

  - Declare and initialize two variables, one int and one double?
    Both should be initialized to the appropriate form of 5.

  - Give good variable names for identifiers to store
    - the speed of an automobile?
    - an hourly pay rate?
    - the highest score on an exam?

2.2

Input and Output

Input and Output

- A data stream is a sequence of data
  - Typically in the form of characters or numbers
- An input stream is data for the program to use
  - Typically originates
    - at the keyboard
    - at a file
- An output stream is the program’s output
  - Destination is typically
    - the monitor
    - a file

Output using cout

- cout is an output stream sending data to the monitor
- The insertion operator "<<" inserts data into cout
- Example:

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

  - This line sends two items to the monitor
    - The value of number_of_bars
    - The quoted string of characters " candy bars\n"
      - Notice the space before the ‘c’ in candy
      - The ‘\n’ causes a new line to be started following the ‘s’ in bars
    - A new insertion operator is used for each item of output

Examples Using cout

- This produces the same result as the previous sample

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

- Here arithmetic is performed in the cout statement

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

- Quoted strings are enclosed in double quotes ("Walter")
  - Don't use two single quotes (')
- A blank space can also be inserted with

```
cout << " " ;
```

  if there are no strings in which a space is desired as in " candy bars\n"

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;
```

Escape Sequences

- Escape sequences tell the compiler to treat characters in a special way
- '\' is the escape character
  - To create a newline in output use

    ```
    \n  –    cout << "\n";
    or the newer alternative
             cout << endl;
    ```

  - Other escape sequences:

\t	-- a tab
\\	-- a backslash character
\"	-- a quote character

Formatting Real Numbers

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

```
double price = 78.5;
cout << "The price is $" << price << endl;
```

  - The output could be any of these:

```
The price is $78.5
The price is $78.500000
The price is $7.850000e01
```

  - The most unlikely output is:

```
The price is $78.50
```

Showing Decimal Places

- cout includes tools to specify the output of type double

- To specify fixed point notation
  - setf(ios::fixed)
- To specify that the decimal point will always be shown
  - setf(ios::showpoint)
- To specify that two decimal places will always be shown
  - precision(2)

- Example:

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

Input Using cin

- cin is an input stream bringing data from the keyboard
- The extraction operator (>>) removes data to be used
- Example:

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

- This code prompts the user to enter data then reads two data items from cin
  - The first value read is stored in number_of_bars
  - The second value read is stored in one_weight
  - Data is separated by spaces when entered

Reading Data From cin

- Multiple data items are separated by spaces
- Data is not read until the enter key is pressed
  - Allows user to make corrections

- Example:

```
                cin >> v1 >> v2 >> v3;
```

  - Requires three space separated values
  - User might type

```
                 34  45  12   <enter key>
```

Designing Input and Output

- Prompt the user for input that is desired
  - cout statements provide instructions

```
cout << "Enter your age: ";
cin >> age;
```

    - Notice the absence of a new line before using cin
- Echo the input by displaying what was read
  - Gives the user a chance to verify data

```
cout << age << " was entered." << endl;
```

Section 2.2 Conclusion

- Can you
  - write an input statement to place a value in the variable the_number?
  - Write the output statement to prompt for the value to store in the_number?
  - Write an output statement that produces a newline?
  - Format output of rational numbers to show 4 decimal places?

2.3

Data Types and Expressions

Data Types and Expressions

- 2 and 2.0 are not the same number
  - A whole number such as 2 is of type int
  - A real number such as 2.0 is of type double

- Numbers of type int are stored as exact values
- Numbers of type double may be stored as approximate values due to limitations on number of significant digits that can be represented

Writing Integer constants

- Type int does not contain decimal points
  - Examples: 34 45 1 89

Writing Double Constants

- Type double can be written in two ways
  - Simple form must include a decimal point
    - Examples: 34.1   23.0034   1.0   89.9

  - Floating Point Notation (Scientific Notation)
    - Examples: 3.41e1   means   34.1
      3.67e17   means
      367000000000000000.0
      5.89e-6   means   0.00000589
  - Number left of e does not require a decimal point
  - Exponent cannot contain a decimal point

Other Number Types

- Various number types have different memory requirements
    - More precision requires more bytes of memory
    - Very large numbers require more bytes of memory
    - Very small numbers require more bytes of memory

Number Types

DISPLAY 2.2 Some Number Types

Type Name	Memory Used	Size Range	Precision
short (also called *short int*)	2 bytes	–32,767 to 32,767	(not applicable)
int	4 bytes	–2,147,483,647 to 2,147,483,647	(not applicable)
long (also called *long int*)	4 bytes	–2,147,483,647 to 2,147,483,647	(not applicable)
float	4 bytes	approximately 10^{-38} to 10^{38}	7 digits
double	8 bytes	approximately 10^{-308} to 10^{308}	15 digits
long double	10 bytes	approximately 10^{-4932} to 10^{4932}	19 digits

These are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system. *Precision* refers to the number of meaningful digits, including digits in front of the decimal point. The ranges for the types *float*, *double*, and *long double* are the ranges for positive numbers. Negative numbers have a similar range, but with a negative sign in front of each number.

Integer types

- long or long int (often 4 bytes)
  - Equivalent forms to declare very large integers

```
long  big_total;
long int big_total;
```

- short or short int (often 2 bytes)
  - Equivalent forms to declare smaller integers

```
short small_total;
short int small_total;
```

Floating point types

- long double (often 10 bytes)
  - Declares floating point numbers with up to 19 significant digits

```
long double big_number;
```

- float (often 4 bytes)
  - Declares floating point numbers with up to 7 significant digits

```
float not_so_big_number;
```

Type char

- Computers process character data too
- char
  - Short for character
  - Can be any single character from the keyboard
- To declare a variable of type char:

```
char letter;
```

char constants

- Character constants are enclosed in single quotes

```
char letter = 'a';
```

- Strings of characters, even if only one character is enclosed in double quotes
  - "a" is a string of characters containing one character
  - 'a' is a value of type character

C++11 Types

- The size of numeric data types can vary from one machine to another.
  - For example, an int might be 32 bits or 64 bits

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

- C++11 also has an “auto” type that deduces the type of the variable based on the expression on the right hand side of the assignment

```
auto x = 2.4 * 10;    // x becomes a double due to 2.4
```

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	

Reading Character Data

- cin skips blanks and line breaks looking for data
- The following reads two characters but skips any space that might be between

```
char symbol1, symbol2;
cin  >>  symbol1  >>  symbol2;
```

- User normally separate data items by spaces

```
J    D
```

- Results are the same if the data is not separated by spaces

```
JD
```

The type *char*

```
#include <iostream>
using namespace std;
int main( )
{
    char symbol1, symbol2, symbol3;

    cout << "Enter two initials, without any periods:\n";
    cin >> symbol1 >> symbol2;

    cout << "The two initials are:\n";
    cout << symbol1 << symbol2 << endl;

    cout << "Once more with a space:\n";
    symbol3 = ' ';
    cout << symbol1 << symbol3 << symbol2 << endl;

    cout << "That's all.";

    return 0;
}
```

Sample Dialogue

```
Enter two initials, without any periods:
J B
The two initials are:
JB
Once more with a space:
J B
That's all.
```

Type string

- string is a class, different from the primitive data types discussed so far
  - Difference is discussed in Chapter 8
  - 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";
```

The string class

```cpp
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string middle_name, pet_name;
    string alter_ego_name;

    cout << "Enter your middle name and the name of your pet.\n";
    cin >> middle_name;
    cin >> pet_name;

    alter_ego_name = pet_name + " " + middle_name;

    cout << "The name of your alter ego is ";
    cout << alter_ego_name << "." << 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

- In general store values in variables of the same type
  - This is a type mismatch:

```
int int_variable;
int_variable = 2.99;
```

  - If your compiler allows this, int_variable will most likely contain the value 2, not 2.99

int ←→ double (part 1)

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

```
int int_variable;
double double_variable;
double_variable = 2.00;
 int_variable = double_variable;
```

  - If allowed, int_variable contains 2, not 2.00

int ←→ double (part 2)

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

```
double double_variable;
double_variable = 2;
```

  - double_variable will contain 2.0

char ← → int

- The following actions are possible but generally not recommended!
- It is possible to store char values in integer variables

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

  value will contain an integer representing 'A'

- It is possible to store int values in char variables

```
char letter = 65;
```

bool ← → int

- The following actions are possible but generally not recommended!
- Values of type bool can be assigned to int variables
  - True is stored as 1
  - False is stored as 0
- Values of type int can be assigned to bool variables
  - Any non-zero integer is stored as true
  - Zero is stored as false

Arithmetic

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

- Example: storing a product in the variable total_weight

```
total_weight  =  one_weight * number_of_bars;
```

Results of Operators

- Arithmetic operators can be used with any numeric type
- An operand is a number or variable used by the operator
- Result of an operator depends on the types of operands
  - If both operands are int, the result is int
  - If one or both operands are double, the result is double

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
    (true for variables or numeric constants)

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

  - The value of quotient is 1, not 1.666…
  - Integer division does not round the result, the fractional part is discarded!

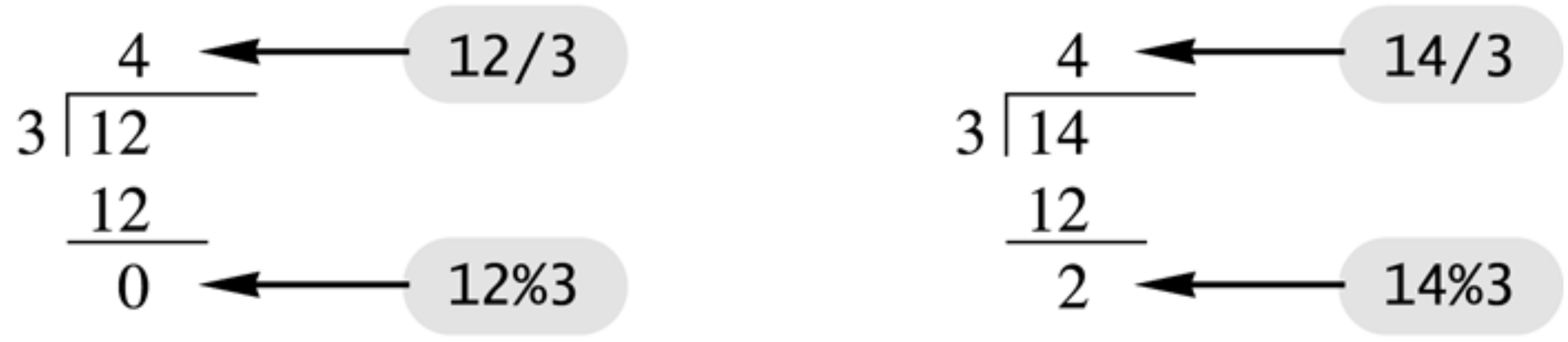
Integer Remainders

- % operator gives the remainder from integer division

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

The value of remainder is 2

Integer Division



Arithmetic Expressions

- Use spacing to make expressions readable
  - Which is easier to read?

```
x+y*z      or    x + y * z
```

- Precedence rules for operators are the same as used in your algebra classes
- Use parentheses to alter the order of operations

```
 x + y * z     ( y is multiplied by z first)
(x + y) * z   ( x and y are added first)
```

Arithmetic Expressions

Mathematical Formula	C++ Expression
$b^2 - 4ac$	`b*b - 4*a*c`
$x(y + z)$	`x*(y + z)`
$\frac{1}{x^2 + x + 3}$	`1/(x*x + x + 3)`
$\frac{a + b}{c - d}$	`(a + b)/(c - d)`

Operator Shorthand

- Some expressions occur so often that C++ contains to shorthand operators for them
- All arithmetic operators can be used this way
  - += count = count + 2; becomes
    count += 2;
  - *= bonus = bonus * 2; becomes
    bonus *= 2;
  - /= time = time / rush_factor; becomes
    time /= rush_factor;
  - %= remainder = remainder % (cnt1+ cnt2); becomes
    remainder %= (cnt1 + cnt2);

2.4

Simple Flow of Control

Simple Flow of Control

- Flow of control
  - The order in which statements are executed
- Branch
  - Lets program choose between two alternatives

Branch Example

- To calculate hourly wages there are two choices
  - Regular time (up to 40 hours)
    - gross_pay = rate * hours;
  - Overtime (over 40 hours)
    - gross_pay = rate * 40 + 1.5 * rate * (hours - 40);
  - The program must choose which of these expressions to use

Designing the Branch

- Decide if (hours >40) is true
  - If it is true, then use
    gross_pay = rate * 40 + 1.5 * rate * (hours - 40);
  - If it is not true, then use
    gross_pay = rate * hours;

Implementing the Branch

- if-else statement is used in C++ to perform a branch

```
if (hours > 40)
   gross_pay  =  rate * 40 + 1.5 * rate * (hours - 40);
else
   gross_pay = rate * hours;
```

An *if-else* Statement

```
#include <iostream>
using namespace std;
int main( )
{
    int hours;
    double gross_pay, rate;

    cout << "Enter the hourly rate of pay: $";
    cin >> rate;
    cout << "Enter the number of hours worked,\n"
         << "rounded to a whole number of hours: ";
    cin >> hours;

    if (hours > 40)
        gross_pay = rate*40 + 1.5*rate*(hours - 40);
    else
        gross_pay = rate*hours;

    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << "Hours = " << hours << endl;
    cout << "Hourly pay rate = $" << rate << endl;
    cout << "Gross pay = $" << gross_pay << endl;

    return 0;
}
```

Sample Dialogue 1

```
Enter the hourly rate of pay: $20.00
Enter the number of hours worked,
rounded to a whole number of hours: 30
Hours = 30
Hourly pay rate = $20.00
Gross pay = $600.00
```

Sample Dialogue 2

```
Enter the hourly rate of pay: $10.00
Enter the number of hours worked,
rounded to a whole number of hours: 41
Hours = 41
Hourly pay rate = $10.00
Gross pay = $415.00
```

Syntax for an `if-else` Statement

A Single Statement for Each Alternative:

```
if (Boolean_Expression)
    Yes_Statement
else
    No_Statement
```

A Sequence of Statements for Each Alternative:

```
if (Boolean_Expression)
{
    Yes_Statement_1
    Yes_Statement_2
      . . .
    Yes_Statement_Last
}
else
{
    No_Statement_1
    No_Statement_2
       . . .
    No_Statement_Last
}
```

Boolean Expressions

- Boolean expressions are expressions that are either true or false
- comparison operators such as '>' (greater than) are used to compare variables and/or numbers
  - (hours > 40)   Including the parentheses, is the boolean expression from the wages example
  - A few of the comparison operators that use two symbols (No spaces allowed between the symbols!)
    - >=   greater than or equal to
    - !=   not equal or inequality
    - = =   equal or equivalent

Comparison Operators

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

if-else Flow Control (1)

-
  ```
  if (boolean expression)
      true statement
  else
      false statement
  ```
- When the boolean expression is true
  - Only the true statement is executed
- When the boolean expression is false
  - Only the false statement is executed

if-else Flow Control (2)

-
  ```
  if (boolean expression)
       {
               true statements
        }
  else
      {
               false statements
       }
  ```
- When the boolean expression is true
  - Only the true statements enclosed in { } are executed
- When the boolean expression is false
  - Only the false statements enclosed in { } are executed

AND

- Boolean expressions can be combined into more complex expressions with
  - && -- The AND operator
    - True if both expressions are true
- Syntax: (Comparison_1) && (Comparison_2)
- Example: if ((2 < x) && (x < 7))
  - True only if x is between 2 and 7
  - Inside parentheses are optional but enhance meaning

OR

- | | -- The OR operator (no space!)
  - True if either or both expressions are true
- Syntax: (Comparison_1) | | (Comparison_2)
- Example: if ((x = = 1) | | (x = = y))
  - True if x contains 1
  - True if x contains the same value as y
  - True if both comparisons are true

NOT

- ! -- negates any boolean expression
  - !(x < y)
    - True if x is NOT less than y
  - !(x = = y)
    - True if x is NOT equal to y
- ! Operator can make expressions difficult to understand…use only when appropriate

Inequalities

- Be careful translating inequalities to C++
- if x < y < z   translates as

```
if ( ( x < y )  && ( y < z ) )
```

NOT

```
if ( x < y < z )
```

Pitfall: Using = or ==

- ' = ' is the assignment operator
  - Used to assign values to variables
  - Example: x = 3;
- '= = ' is the equality operator
  - Used to compare values
  - Example: if (x == 3)
- The compiler will accept this error:

  if (x = 3)

  but stores 3 in x instead of comparing x and 3
  - Since the result is 3 (non-zero), the expression is true

Compound Statements

- A compound statement is more than one statement enclosed in { }
- Branches of if-else statements often need to execute more that one statement
- Example:

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

Compound Statements Used with *if-else*

```
if (my_score > your_score)
{
    cout << "I win!\n";
    wager = wager + 100;
}
else
{
    cout << "I wish these were golf scores.\n";
    wager = 0;
}
```

Branches Conclusion

- Can you
  - Write an if-else statement that outputs the word High if the value of the variable score is greater than 100 and Low if the value of score is at most 100? The variables are of type int.
  - Write an if-else statement that outputs the word Warning provided that either the value of the variable temperature is greater than or equal to 100, or the of the variable pressure is greater than or equal to 200, or both. Otherwise, the if_else sttement outputs the word OK. The variables are of type int.

Simple Loops

- When an action must be repeated, a loop is used
- C++ includes several ways to create loops
- We start with the while-loop
- Example:

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

- Output: Hello Hello Hello
  when count_down starts at 3

A *while* Loop

```
#include <iostream>
using namespace std;
int main( )
{
    int count_down;

    cout << "How many greetings do you want? ";
    cin >> count_down;

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

    cout << endl;
    cout << "That's all!\n";

    return 0;
}
```

Sample Dialogue 1

```
How many greetings do you want? 3
Hello Hello Hello
That's all!
```

Sample Dialogue 2

```
How many greetings do you want? 1
Hello
That's all!
```

Sample Dialogue 3

```
How many greetings do you want? 0

That's all!
```

The loop body is executed zero times.

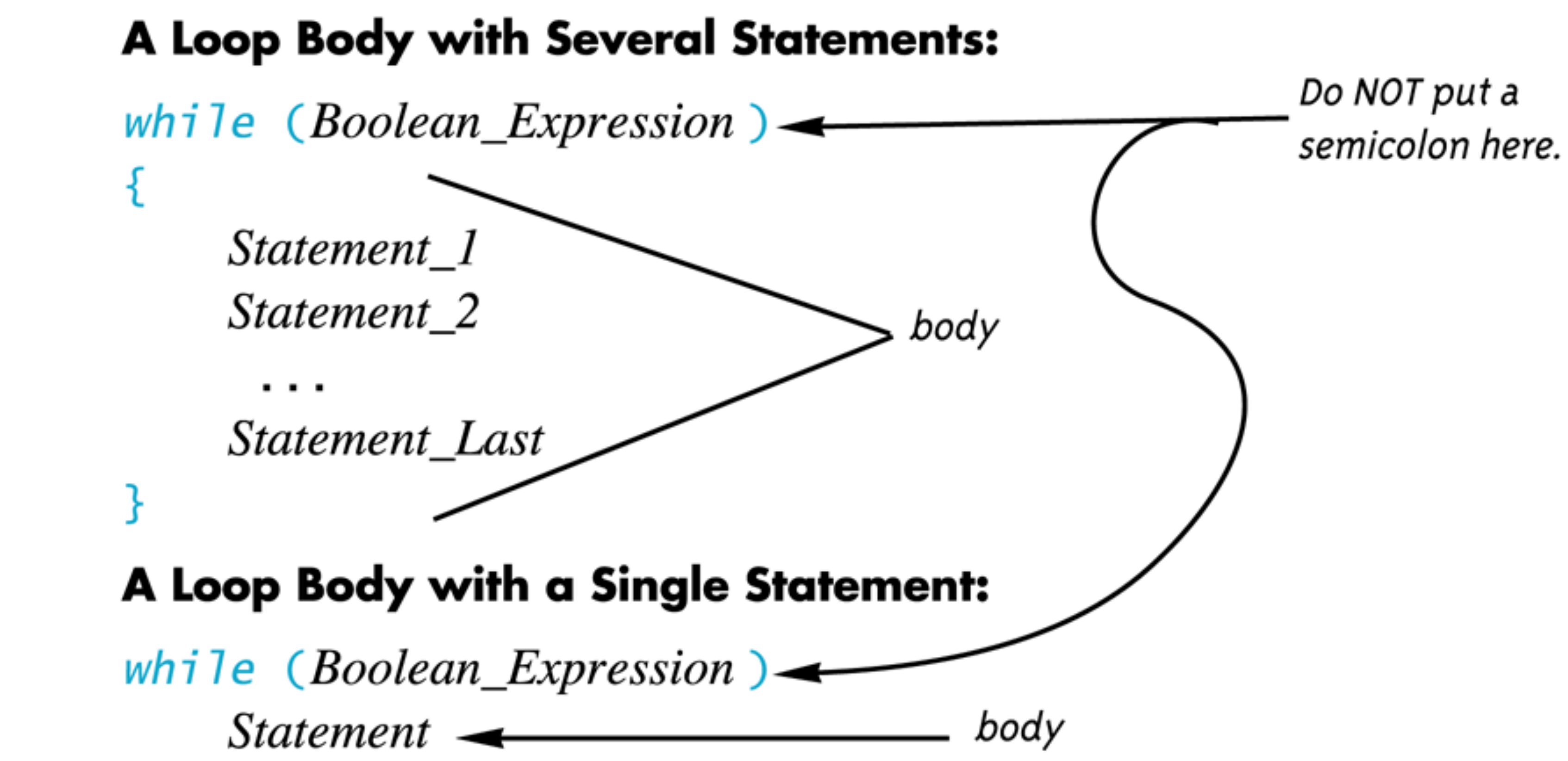
While Loop Operation

- First, the boolean expression is evaluated
  - If false, the program skips to the line following the while loop
  - If true, the body of the loop is executed
    - During execution, some item from the boolean expression is changed
  - After executing the loop body, the boolean expression is checked again repeating the process until the expression becomes false
- A while loop might not execute at all if the boolean expression is false on the first check

while Loop Syntax

- while (boolean expression is true)
```
      {
              statements to repeat
       }
```
  - Semi-colons are used only to end the statements
    within the loop
- while (boolean expression is true)
```
          statement to repeat
```

Syntax of the *while* Statement



A Loop Body with Several Statements:

```
while (Boolean_Expression )
{
      Statement_1
      Statement_2
        . . .
      Statement_Last
}
```

A Loop Body with a Single Statement:

```
while (Boolean_Expression )
      Statement
```

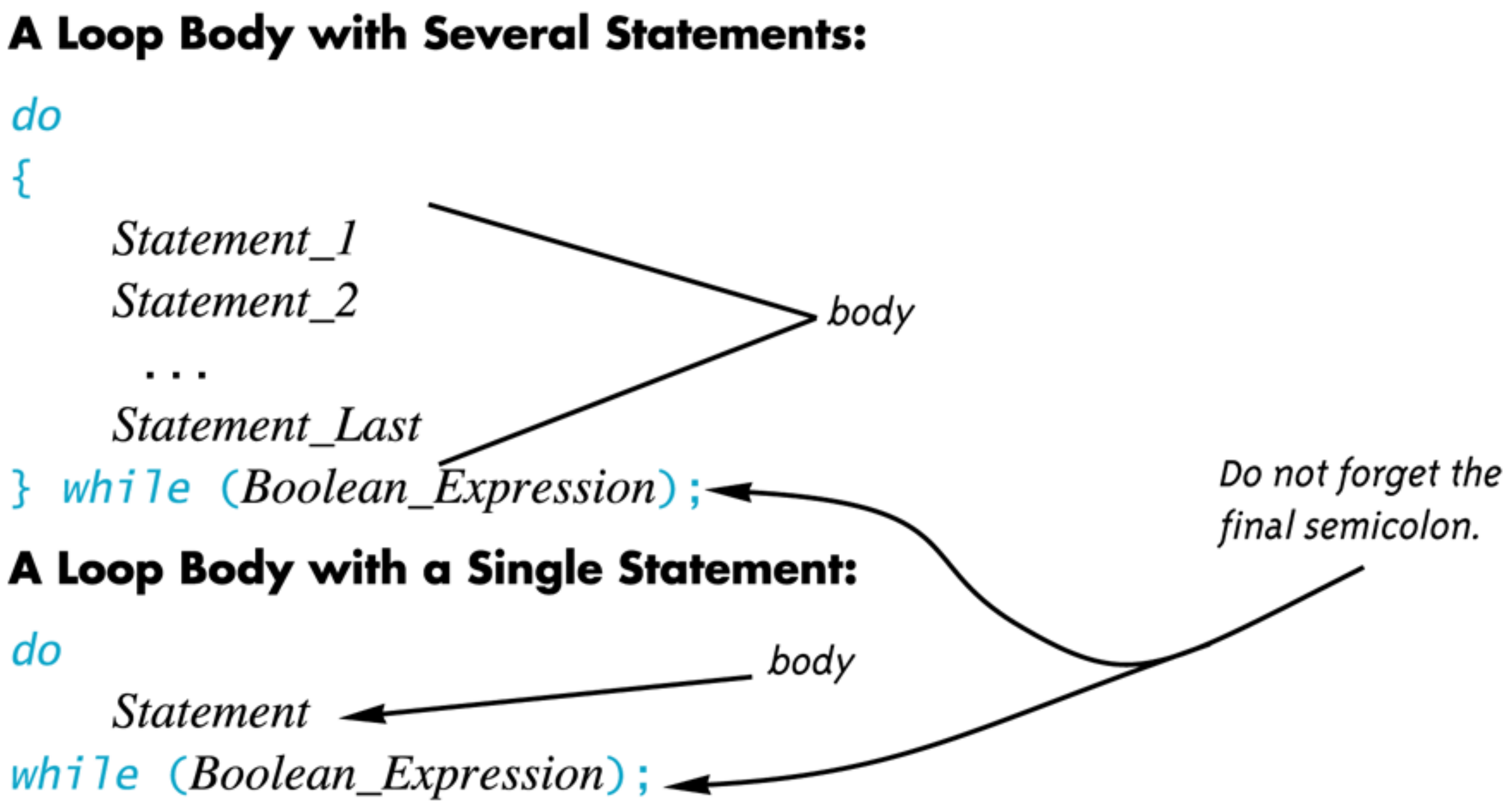
do-while loop

- 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);
```

Syntax of the *do-while* Statement



A Loop Body with Several Statements:

```
do
{
      Statement_1
      Statement_2
        . . .
      Statement_Last
} while (Boolean_Expression);
```

A Loop Body with a Single Statement:

```
do
      Statement
while (Boolean_Expression);
```

A *do-while* Loop

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

    do
    {
        cout << "Hello\n";
        cout << "Do you want another greeting?\n"
             << "Press y for yes, n for no,\n"
             << "and then press return: ";
        cin >> ans;
    } while (ans == 'y' || ans == 'Y');

    cout << "Good-Bye\n";

    return 0;
}
```

Sample Dialogue

```
Hello
Do you want another greeting?
Press y for yes, n for no,
and then press return: y
Hello
Do you want another greeting?
Press y for yes, n for no,
and then press return: Y
Hello
Do you want another greeting?
Press y for yes, n for no,
and then press return: n
Good-Bye
```

Increment/Decrement

- Unary operators require only one operand
  - + in front of a number such as +5
  - - in front of a number such as -5
- ++ increment operator
  - Adds 1 to the value of a variable

    ```
                        x ++;
    is equivalent to    x = x + 1;
    ```
- -- decrement operator
  - Subtracts 1 from the value of a variable

    ```
                        x --;
    is equivalent to    x = x – 1;
    ```

Sample Program

- Bank charge card balance of $50
- 2% per month interest
- How many months without payments before your balance exceeds $100
- After 1 month: $50 + 2% of $50 = $51
- After 2 months: $51 + 2% of $51 = $52.02
- After 3 months: $52.02 + 2% of $52.02 …

Charge Card Program

```
#include <iostream>
using namespace std;
int main( )
{
    double balance = 50.00;
    int count = 0;

    cout << "This program tells you how long it takes\n"
         << "to accumulate a debt of $100, starting with\n"
         << "an initial balance of $50 owed.\n"
         << "The interest rate is 2% per month.\n";

    while (balance < 100.00)
    {
        balance = balance + 0.02 * balance;
        count++;
    }

    cout << "After " << count << " months,\n";
    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << "your balance due will be $" << balance << endl;

    return 0;
}
```

Sample Dialogue

```
This program tells you how long it takes
to accumulate a debt of $100, starting with
an initial balance of $50 owed.
The interest rate is 2% per month.
After 36 months,
your balance due will be $101.99
```

Infinite Loops

- Loops that never stop are infinite loops
- The loop body should contain a line that will eventually cause the boolean expression to become false
- Example: Print the odd numbers less than 12

```
x = 1;
while (x != 12)
    {
        cout << x << endl;
        x = x + 2;
    }
```

- Better to use this comparison: while (x < 12)

Section 2.4 Conclusion

- Can you
  - Show the output of this code if x is of type int?

```
x = 10;
while ( x > 0)
  {
         cout << x << endl;
         x = x – 3;
   }
```

  - Show the output of the previous code using the comparison x < 0 instead of x > 0?

2.5

Program Style

Program Style

- A program written with attention to style
  - is easier to read
  - easier to correct
  - easier to change

Program Style - Indenting

- Items considered a group should look like a group
  - Skip lines between logical groups of statements
  - Indent statements within statements

```
if (x = = 0)
    statement;
```

- Braces {} create groups
  - Indent within braces to make the group clear
  - Braces placed on separate lines are easier to locate

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
    gross_pay = 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 have no mnemonic value
- 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

Constants

- 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

Comments and Named Constants

```
//File Name: health.cpp (Your system may require some suffix other than cpp.)
//Author: Your Name Goes Here.
//Email Address: you@yourmachine.bla.bla
//Assignment Number: 2
//Description: Program to determine if the user is ill.
//Last Changed: September 23, 2004

#include <iostream>
using namespace std;
int main( )
{
    const double NORMAL = 98.6;//degrees Fahrenheit
    double temperature;

    cout << "Enter your temperature: ";
    cin >> temperature;

    if (temperature > NORMAL)
    {
        cout << "You have a fever.\n";
        cout << "Drink lots of liquids and get to bed.\n";
    }
    else
    {
        cout << "You don't have a fever.\n";
        cout << "Go study.\n";
    }

    return 0;
}
```

Your programs should always begin with a comment similar to this one.

Sample Dialogue

```
Enter your temperature: 98.6
You don't have a fever.
Go study.
```

Section 2.5 Conclusion

- Can you
  - Create a named constant of type double?
  - Determine if a program can modify the value of a constant?
  - Describe the benefits of comments?
  - Explain why indenting is important in a program?
  - Explain why blank lines are important in a program?

Chapter 2 -- End