Chapter 2

C++ Basics

Prof. Chien-Nan (Jimmy) Liu
Dept. of Electrical Engineering
National Chiao-Tung Univ.

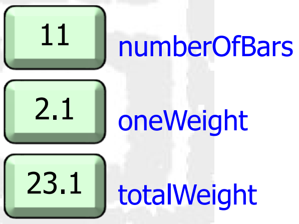
Tel: (03)5712121 ext:31211
E-mail: jimmyliu@nctu.edu.tw
http://mseda.ee.nctu.edu.tw/jimmyliu

Overview

- *2.1 Variables and Assignments*
- 2.2 Input and Output
- 2.3 Data Types and Expressions
- 2.4 Simple Flow of Control
- 2.5 Program Style

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
  - However, we cannot erase the memory location
    - Some value is always there



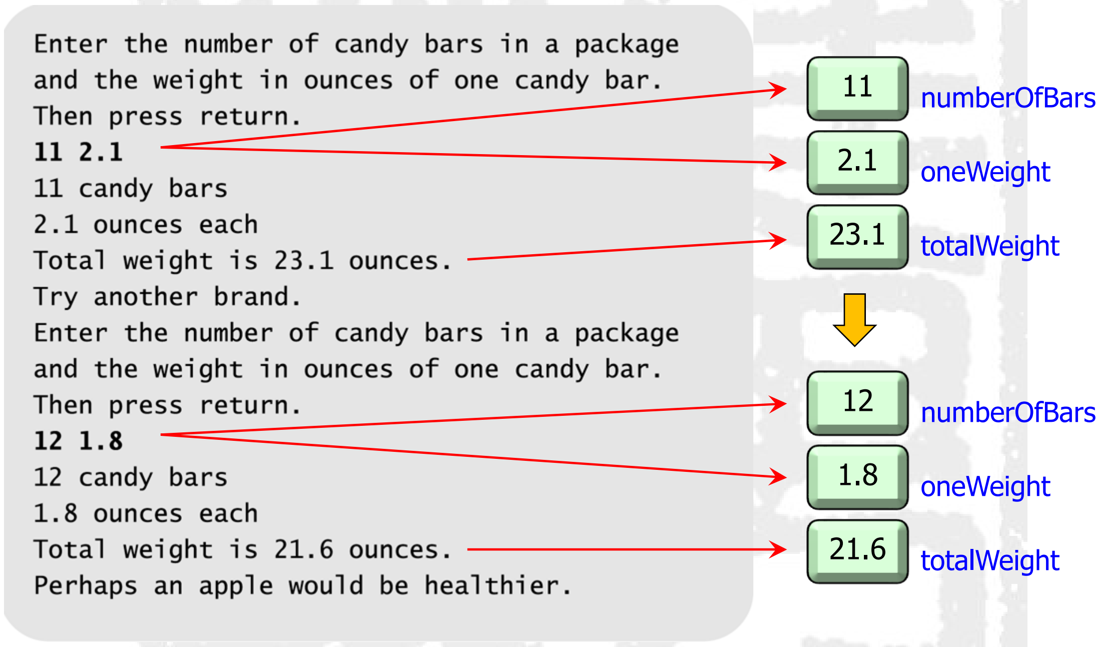
A C++ Program

variables

```
#include <iostream>
using namespace std;
int main( )
{
  int numberOfBars;
  double oneWeight, totalWeight;

  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 >> numberOfBars;
  cin >> oneWeight;

  totalWeight = oneWeight * numberOfBars;

  cout << numberOfBars << " candy bars\n";
  cout << oneWeight << " ounces each\n";
  cout << "Total weight is " << totalWeight << "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 >> numberOfBars;
  cin >> oneWeight;
  totalWeight = oneWeight * numberOfBars;

  cout << numberOfBars << " candy bars\n";
  cout << oneWeight << " ounces each\n";
  cout << "Total weight is " << totalWeight << " ounces.\n";

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

  return 0;
}
```

assignment

Sample Dialogue



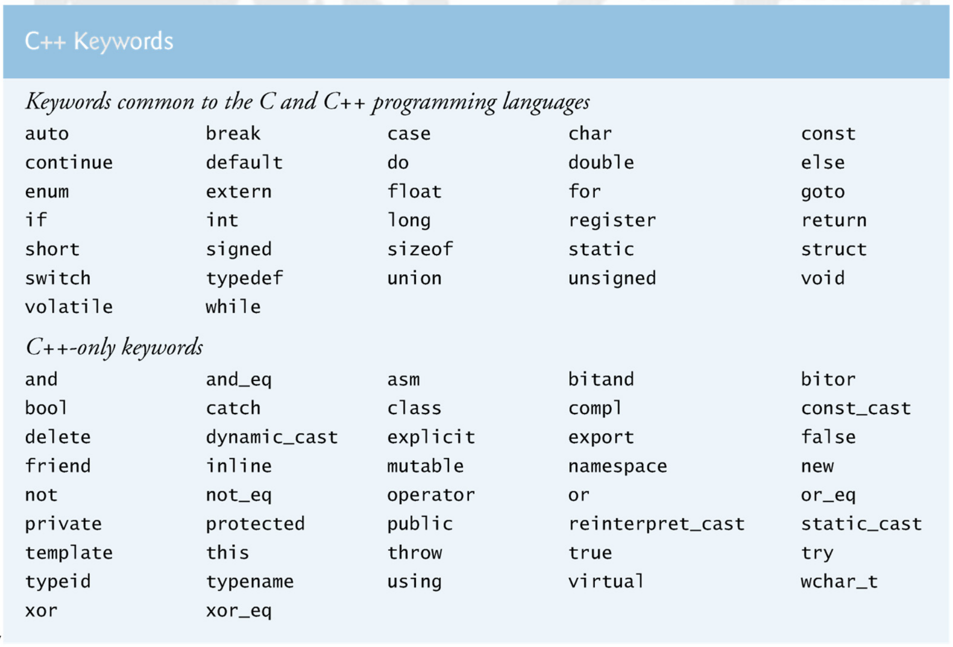
```
Enter the number of candy bars in a package
and the weight in ounces of one candy bar.
Then press return.
11 2.1
11 candy bars
2.1 ounces each
Total weight is 23.1 ounces.
Try another brand.
Enter the number of candy bars in a package
and the weight in ounces of one candy bar.
Then press return.
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 your data
    - Not too long (ex: ThisIsVariable) because of typing efforts
  - First character must be
    - a letter (ex: a, B, c, D …)
    - the underscore (_) character
  - Remaining characters must be
    - letters
    - numbers (ex: 1, 2, 3, …)
    - underscore character
- Identifiers are case-sensitive !! (c1 vs C1?)

Keywords Cannot be Identifiers

- Keywords (also called reserved words)
  - Cannot be identifiers because they have special meanings



C++ Keywords

Keywords common to the C and C++ programming languages

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

C++-only keywords

and	and_eq	asm	bitand	bitor
bool	catch	class	compl	const_cast
delete	dynamic_cast	explicit	export	false
friend	inline	mutable	namespace	new
not	not_eq	operator	or	or_eq
private	protected	public	reinterpret_cast	static_cast
template	this	throw	true	try
typeid	typename	using	virtual	wchar_t
xor	xor_eq			

Declaring Variables (1/2)

- Before use, declare variables to reserve space
  - Tells the compiler the type of data to store
  - Declaration syntax:

Type_name *Variable_1*, *Variable_2*, . . . ;

Don't forget semi-colon !!

- Examples: int *numberOfBars*;

  double *one_weight*, *totalWeight*;

  - int is an abbreviation for integer
    - could store 3, 102, 3211, -456, etc.
    - *numberOfBars* is of type integer
  - double represents real numbers (with fractional part)
    - could store 1.34, 4.0, -345.6, etc.
    - *one_weight* and *totalWeight* are both of type double

Declaring Variables (2/2)

Two locations for variable declarations

- At the beginning

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

- Immediately prior to use

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

Assignment Statements

totalWeight = oneWeight + numberOfBars;

- An assignment changes the value of a variable
  - *totalWeight* is set to the sum *oneWeight* + *numberOfBars*
  - 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 -- myCost = yourCost;
    - Expressions -- circumference = diameter * 3.14159;
- The '=' operator in C++ is not an equal sign
  - C = 3 (C becomes 3) v.s. C == 3 (is C equal to 3)

Initializing Variables

- Declaring a variable does not give it a value
  - Reserve a space for you, but nobody there …
- Initializing variables help to avoid unpredicted results
- Variables can be initialized in assignment statements

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

- Declaration and initialization can be combined
  - Method 1

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

  - Method 2

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

without initial value

Overview

- 2.1 Variables and Assignments
- *2.2 Input and Output*
- 2.3 Data Types and Expressions
- 2.4 Simple Flow of Control
- 2.5 Program Style

Input and Output Streams

- A data stream is a sequence of data
  - Typically in the form of characters or numbers
- An input stream gets the required data from
  - the keyboard
  - a file
- An output stream sends the program's output to
  - the monitor
  - a file
- Include Directives add library files to our programs
  - Ex: #include <iostream> makes *cin* and *cout* available
- Using Directives to include a collection of defined names
  - To use them in standard way → using namespace std;

Output Using cout

- The insertion operator "<<" inserts data into *cout*
  - It indicates the data flow, not "smaller than"
- Example:

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

  - This line sends two items to the monitor
    - The **value** of numberOfBars
    - The quoted **string** of characters " candy bars\n"
      - The space before the 'c' in the string also appears in monitor
    - The '**\n**' causes a new line following the 's' in bars
  - An insertion operator is used for each item of output
  - Quoted strings are enclosed in double quotes ("candy")
    - Don't use two single quotes (') → single character only

Examples Using cout

- This produces the same result as the previous sample

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

- Arithmetic operation is allowed in the cout statement

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

  - Output like this way → Total cost is $25 (assume price=20, tax=5)
  - Complete the arithmetic operation first then print the result
- A blank space can also be inserted separately

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

without space

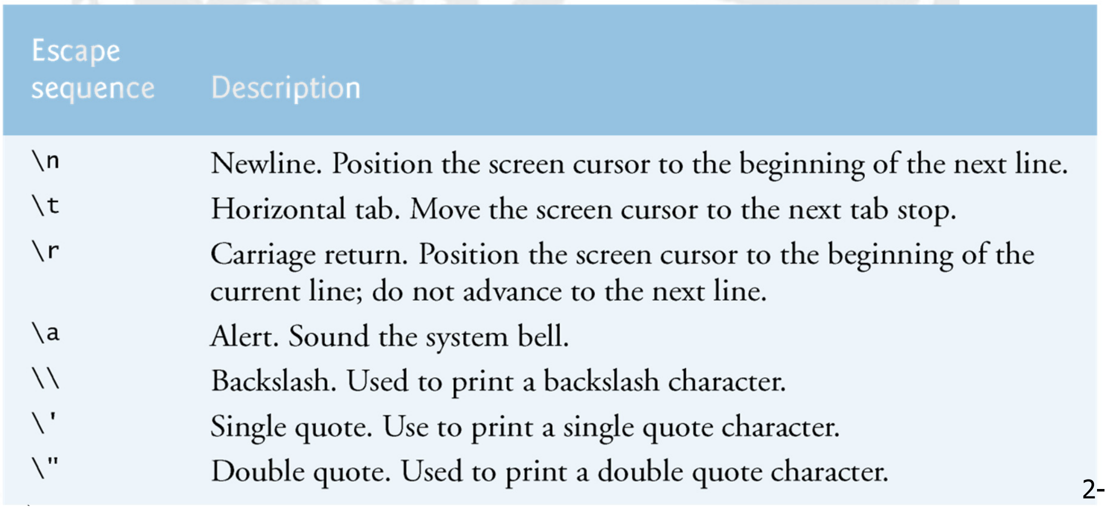
  - Ex: cout << "Numbers:" << a << b → Numbers:78

    cout << "Numbers: " << a << " " << b → Numbers: 7 8

space space

Escape Sequences

- Back slash ('\') tells the compiler to treat the next character in a special way → escape sequence
- Ex: **\n** moves the cursor to next line → cout << "\n";
  - An alternative way is cout << endl; (a special command in iostream)



Escape sequence	Description
`\n`	Newline. Position the screen cursor to the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
`\a`	Alert. Sound the system bell.
`\\`	Backslash. Used to print a backslash character.
`\'`	Single quote. Use to print a single quote character.
`\"`	Double quote. Used to print a double quote character.

Formatting Real Numbers

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

```
cout << "The price is $" << price << endl;
```

  - The output could be any of these: (assume price = 78.5)

```
The price is $78.5
The price is $78.500000
The price is $7.85e01
```

scientific notation:
7.85e01 = 7.85×10^1
7.85e-2 = 7.85×10^{-2}

- cout includes tools to specify the output of type double
  - setf(ios::fixed) → specify fixed point notation (ex: 78.5)
  - setf(ios::scientific) → scientific notation (ex: 7.85e01)
  - setf(ios::showpoint) → always show decimal point (75->75.0)
  - precision(2) → two decimal places are shown (ex: 78.50)
- Ex: cout.setf(ios::fixed); cout.precision(2); cout << …

Input Using cin

- The extraction operator (>>) brings data from keyboard
  - It indicates the data flow, not “larger than”
- Example:

```
cout << "Enter the number of bars in a package\n";
cin >> numberOfBars;
```

  - Prompt the user to enter data then read an item from *cin*
  - The first value read is stored in *numberOfBars*
- 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;`
  - User might type → 34 45 12 <enter key>
  - After cin, v1 = 34, v2 = 45, v3 = 12

Designing Input and Output

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

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

    - You can decide if a new line is required before using *cin*
  - Otherwise, the program is "hanged" to wait for input
- Check the obtained data to prevent invalid input
- Echo the input by displaying what was read
  - Gives the user a chance to verify data

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

Overview

- 2.1 Variables and Assignments
- 2.2 Input and Output
- *2.3 Data Types and Expressions*
- 2.4 Simple Flow of Control
- 2.5 Program Style

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
  - Ex: 34 45 1 89 (no decimal points)
- Numbers of type *double* are stored as approximate values due to format limitations (IEEE 754 standard)
  - Ex: 23.034 3.67e17 0.002 5.89e-6
- Various data types require different memory spaces
  - More precision requires more bytes of memory
  - Very large/small numbers require more bytes of memory

Some Number Types

	Type Name	Memory Used	Size Range	Precision
小 →	*short* (also called *short int*)	2 bytes	-32,768 to 32,767	(not applicable)
中 →	*int*	4 bytes	-2,147,483,648 to 2,147,483,647	(not applicable)
大 →	*long* (also called *long int*)	4 bytes	-2,147,483,648 to 2,147,483,647	(not applicable)
小 →	*float*	4 bytes	approximately 10^{-38} to 10^{38}	7 digits
中 →	*double*	8 bytes	approximately 10^{-308} to 10^{308}	15 digits
大 →	*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.*

More on Data Types

- Integer types:
  - long or long int (often 4 bytes) → for large integers (>65535)
  - short or short int (often 2 bytes) → for small integers
- Floating-point types:
  - long double (often 10 bytes) → up to 19 significant digits
  - float (often 4 bytes) → only 7 significant digits
- Character types:
  - char (often 1 byte) → a single character from the keyboard
  - Character constants are enclosed in single quotes
    - char letter = 'a'; (O) char letter = "a"; (X) → a string of characters
- Boolean types:
  - bool (often 1 byte) → result of logical event (TRUE or FALSE)

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 the size and whether the data is signed or unsigned
  - Ex: *int8_t* (signed 8-bit integer), *uint16_t* (unsigned 16-bit integer)
- 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

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 Characters and Strings

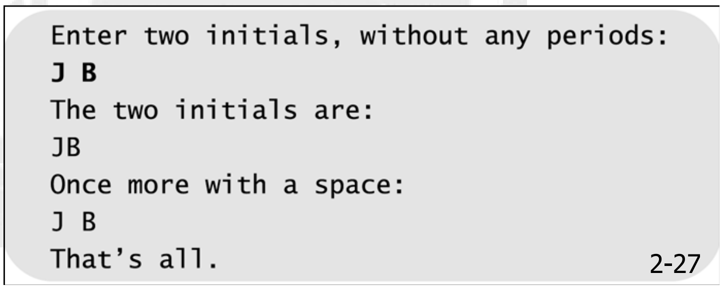
- *cin* normally skips blanks and stops at endline
  - However, *char* type only allows ONE character
- Ex: the following code reads two characters

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

  - User normally separate data items by spaces
    J D → symbol1 = 'J', symbol2 = 'D'
  - Results are the same if the data is not separated by spaces
    JD → symbol1 = 'J', symbol2 = 'D'
- String can have multiple characters in it
  - If the data is not separated by spaces, it becomes a single string
    JD → string1 = "JD"
- Type *string* is a special class defined in library <string>
  - #include <string>

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;
   cout << symbol2 << endl;
   cout << "That's all.";

   return 0;
}
```

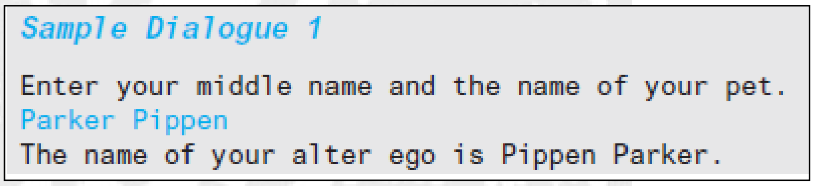


```
Enter two initials, without any periods:
J B
The two initials are:
JB
Once more with a space:
J B
That's all.
```

The string Class

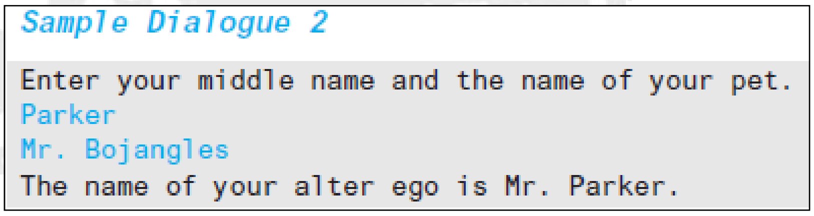
```
#include <iostream>
#include <string>
using namespace std;
int main()
{
   string middleName, petName;
   string alterEgoName;

   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;
}
```

$\longrightarrow$ concatenate those strings



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 Conversion Rules

- In general, the stored values in variables should match the specified types
  - This is a type mismatch: `int intVariable = 2.99;`
- C++ converts the values to new type automatically, but the values may be modified during conversion
  - Especially for the conversion from "large" type to "small" type
  - Ex: *intVariable* will most likely contain the value 2, not 2.99
- For mixed-type expressions, C++ promotes each value to the "highest" type in the expression
  - A temporary version of each value is created and used for the expression — the original values remain unchanged

Examples of Type Conversion

- int → double
  ```
  int intVariable = 2;
  double doubleVariable = intVariable;  → contains 2.00
                                          (lossless conversion)
  ```
- double → int
  ```
  double doubleVariable = 2.99;
  int intVariable = doubleVariable;       → contains 2, not 2.99
                                          (skip all fractional part)
  ```
- char ←→ int
  - `int value = 'A';` → store an integer code for 'A' (ASCII code)
  - `char letter = 65;` → allowed, but treated as 'A'
  - Such confusing operations are generally not recommended!
- bool ←→ int
  - bool → int: FALSE is stored as 0; TRUE is stored as 1
  - int → bool: zero is stored as FALSE; non-zero is stored as TRUE

Arithmetic

- Arithmetic is performed with operators
  - + for addition
  - - for subtraction
  - * for multiplication
  - / for division
  - % for modulus (remainder)
- Ex: totalWeight = oneWeight * numberOfBars;
  - Store the product in the variable *totalWeight*
- Arithmetic operators support all numeric types
- Result of an operator depends on the operand types
  - If both operands are int, the result is int
  - If one or both operands are double, the result is double

Results of Division

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

```
double divisor, dividend, quotient;
dividend = 5; divisor = 3;
quotient = dividend / divisor;   → quotient = 1.66…
```

- int / int produces an integer result (true for variables or numeric constants)

```
int divisor, dividend, quotient;
dividend = 5; divisor = 3;
quotient = dividend / divisor;   → quotient = 1, not 1.66...
```

  - Integer division does not round the result, the fractional part is discarded!

Pitfall: Whole Numbers in Division

- Suppose you are an architect who charges $5000 per mile to landscape a highway
  - Given the highway length, there are 2 ways to calculate price
- 1st approach

  totalPrice = 5000 * (feet/5280.0); → floating-point division
  - Given feet=15000, 5000*(15000/5280.0)=14204.5454…
- 2nd approach

  totalPrice = 5000 * (feet/5280); → integer division
  - 5000*(15000/5280)=5000*2=10000
  - Wrong result because of the missing decimal point
- Be careful about the operand type for division!!

Integer Remainders

- % operator gives the remainder from integer division

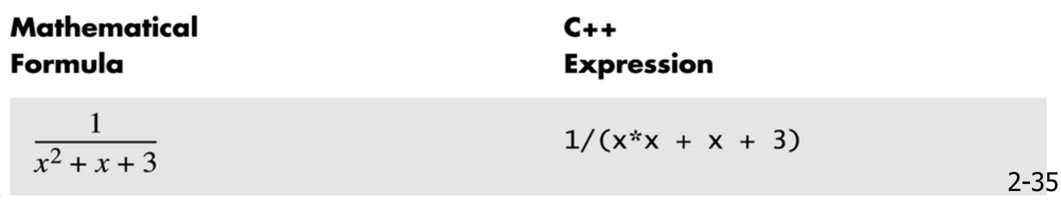
```
int divisor, dividend, remainder;
dividend = 5; divisor = 3;
remainder = dividend % divisor;   → remainder = 2
```
  - To get all results of $\underline{5 \div 3 = 1 \ldots 2}$ require 2 operations
- If one of the operands are double, you will get an error message "invalid operands"
  - Special function is required to get float-point remainder
- Modulus operator is useful to test multiple (倍數) of an integer or form an equal difference (等差) series
  - i % 2 == 0 → i = 0, 2, 4, 6, 8 … (even numbers)
  - i % 3 == 1 → i = 1, 4, 7, 10, … (difference = 3)

Arithmetic Expressions

- Use spacing to make expressions readable
  - x+y*z v.s. x + y * z
- Precedence rules for operators are the same as used in your algebra classes
  - Multiply/division first, then addition/substraction
- 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)
  - If you are not sure about the precedence, add parentheses!!
- EX:



Mathematical Formula	C++ Expression
$\frac{1}{x^2 + x + 3}$	`1/(x*x + x + 3)`

Operator Shorthand

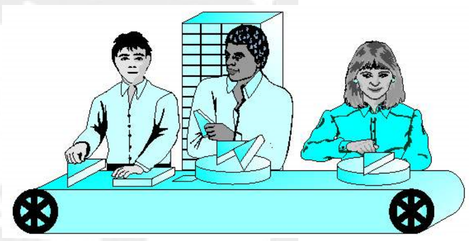
- Some expressions occur so often
  - C++ contains shorthand operators for them
- All arithmetic operators can be used this way
  - += count = count + 2; becomes
    count += 2;
  - *= bonus = bonus * 2; becomes
    bonus *= 2;
  - /= time = time / rushFactor; becomes
    time /= rushFactor;
  - %= remainder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

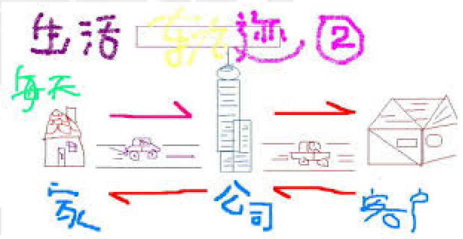
Overview

- 2.1 Variables and Assignments
- 2.2 Input and Output
- 2.3 Data Types and Expressions
- *2.4 Simple Flow of Control*
- 2.5 Program Style

Flow of Control in C++

- Normally, statements in a program execute one after the other
  - Called sequential execution
- Flow of control
  - Specify the next executing statement that is not the next one in sequence
- Branch
  - Let program choose between two (or more) alternatives
- Repetition
  - Perform some statements repeatedly
  - Also called looping statement





Selection Structures

- C++ provides three types of selection statements (discussed in this chapter and Chapter 3)
- The if selection statement: (single selection)
  - The condition is true: perform (selects) the following action
  - The condition is false: skip the action
- The if…else selection statement: (double selection)
  - The condition is true: perform (selects) the following action
  - The condition is false: perform a different action
- The switch selection statement: (multiple selection)
  - Perform one of many different actions, depending on the value of selection expression
  - Introduced in Chapter 3

Repetition Structures

- C++ provides three repetition statements for performing statements repeatedly
  - Also called looping statement
  - These are while, do…while and for statements
- The while and for statements perform the action (or group of actions) in their bodies zero or more times
  - for loop can be viewed as a special case of while loop (introduced in Chapter 3)
- The do…while statement performs the action (or group of actions) in its body at least once

Branch Example

- There are two choices to calculate hourly wages
  - Regular time (up to 40 hours)
    - grossPay = rate * hours;
  - Overtime (over 40 hours)
    - grossPay = 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
    grossPay = rate * 40 + 1.5 * rate * (hours - 40);
  - If it is not true, then use
    grossPay = rate * hours;
- if-else statement is used in C++ to perform a branch

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

An if-else Statement

```
int hours;
double grossPay, 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)
    grossPay = rate*40 + 1.5*rate*(hours - 40);
else
    grossPay = 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 = $" << grossPay << endl;
```

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
    ```
    only ONE statement can be executed!
- A Sequence of Statements for Each Alternative:

  -
    ```
    if (Boolean_Expression)        else
    {                              {
      Yes_Statement_1                No_Statement_1
      Yes_Statement_2                No_Statement_2
       ...                            ...
      Yes_Statement_Last             No_Statement_Last
    }                              }
    ```
  - { } is also called a compound statement

Compound Statements

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

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

Comparison Operators

- The results of Boolean expressions are either true or false
- For two-symbol operators, no space is allowed in between

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

Pitfall: Using = or ==

- ‘ = ’ is the assignment operator
  - Used to assign values to variables
  - Example: x = 3; ⟶ x becomes 3
- ‘ == ’ is the equality operator
  - Used to compare values
  - Example: if (x == 3) ⟶ x may not be 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
  - This is called logical error!!

Combine Boolean Expressions

- Boolean expressions can be combined into more complex expressions with AND/OR/NOT
- && -- The AND operator
  - True if both expressions are true
  - Syntax: (Comparison_1) && (Comparison_2)
- || -- The OR operator
  - True if any one expression is true (or both)
  - Syntax: (Comparison_1) || (Comparison_2)
- Ex:
  - if ((2 < x) && (x < 7)) → true only if x is between 2 and 7
  - if ((x==1) || (x==y)) → true if x is 1 or x equals to y
  - Inside parentheses are optional but enhance understanding

NOT & Inequalities

- ! -- 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
- Be careful translating inequalities to C++
- Ex: if x < y < z , it is translated as
  - if (x < y < z) → wrong!!
  - if ((x < y) && (y < z)) → correct!!
  - Comparison operators only have two operands

Simple Loops – While Loop

- When an action must be repeated, a loop is used
  - We start with the while-loop
- How about printing “Hello” for 3 times?
  - No need to use 3 *cout* statements (see code in next page)
- First, the boolean expression after while is evaluated
  - If FALSE, skip the loop body and jump to the line after loop
  - If TRUE, the loop body is executed
    - Some variables in the boolean expression may be changed
  - After executing the loop body, the boolean expression is checked again to repeat the whole process
    - Stop the loop when the boolean expression becomes false
  - If the expression is initially false, nothing happens at all !!

An Example of while Loop

```
#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;
}
```

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.

Syntax of 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 not put a semicolon here.

body

body

do-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
- Reduce coding efforts in some cases:

```
total = 0; grade = 0;
cout << “Enter grade, ”;
cout << “-1 to end: ”;
cin >> grade;
while (grade != -1) {
  total = total + grade;
  cout << “Enter grade, ”;
  cout << “-1 to end: ”;
  cin >> grade;
}
```

```
total = 0; grade = 0;
do {
  total = total + grade;
  cout << “Enter grade, ”;
  cout << “-1 to end: ”;
  cin >> grade;
} while (grade != -1);
```

Syntax of the do-while Statement

```
// A Loop Body with Several Statements:
do
{
    Statement_1        ⎫
    Statement_2        ⎬ body
     . . .             ⎪
    Statement_Last     ⎭
} while ( Boolean_Expression ); ← Do not forget the final semicolon.

// A Loop Body with a Single Statement:
do
    Statement  ← body
while ( Boolean_Expression ); ← Do not forget the final semicolon.
```

An Example of 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;
}
```

```
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
```

```
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
```

Avoid Infinite Loops

- Loops that never stop are infinite loops
- 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;            12 ??
    x = x + 2;  ——→  x = 1, 3, 5, 7, 9, 11, 13 …
}
```

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

Increment/Decrement

- Unary operators require only one operand
  - Ex: – in front of a number such as –a (change the sign of a)
- ++ increment operator
  - Adds 1 to the value of a variable

    x = x + 1 → x += 1 → x++ (or ++x)
- – – decrement operator
  - Subtracts 1 from the value of a variable

    y = y – 1 → y –=1 → y-- (or --y)
- Postfix: c++ (used first)

  ```
  int c = 5;
  cout << c++;
  ```
  → output is **5**
- Prefix: ++c (increased first)

  ```
  int c = 5;
  cout << ++c;
  ```
  → output is **6**

Example: Charge Card Program

- Initial balance of the bank charge card is $50
- Interest = 2% per month
- How many months are required to make your balance exceed $100 (assume no payment is made)

- 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 …

C++ Code for Charge Card Program

```
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;
```

Overview

- 2.1 Variables and Assignments
- 2.2 Input and Output
- 2.3 Data Types and Expressions
- 2.4 Simple Flow of Control
- *2.5 Program Style*

Program Styles

- A program written with attention to style
  - is easier to read
  - easier to correct
  - easier to change
- “Syntax” and “Style” are different guidelines
  - Syntax – have to follow
  - Style – better to follow
- Try to build up good habit from the beginning
  - Taking the course “software engineering” may help

Program Style - Indenting

- Make items in a group look like a group!!
  - Add empty lines between logical groups of statements
  - Proper indent within statements
    - Any number of spaces is ok, but try to keep each level consistent!!
- Braces {} create groups
  - Indent within braces to make the group clear
  - Braces placed on separate lines are easier to locate

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

    cout << "How many hello? ";
    cin >> count;

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

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

    return 0;
}
```

Program Style - Comments

- // is the symbol for a single line comment
  - Comments are extra notes to help people understand
  - 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
     */
    ```
- Comments are ignored by compiler
  - Insert comments as many as you want

Program Style - Constants

- Number constants used throughout a program are difficult to find and change when needed
- Constants
  - Allow us to give a meaning for the number constant
  - Allow us to change all occurrences simply by changing the value of the constant
- **const** is the keyword to declare a constant
- Ex: `const int WINDOW_COUNT = 10;`
  - Declares a constant named WINDOW_COUNT
  - It cannot be changed during 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.
//Last Changed: September 23, 2017

#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 << "Go study.\n";

   return 0;
}
```

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