Universiti Malaysia PAHANG
Engineering • Technology • Creativity

BTE2313

Chapter 4: Input and output

by
Sulastri Abdul Manap
Faculty of Engineering Technology
sulastri@ump.edu.my

Objectives

- In this chapter, you will learn about:
  1) Learn about input and output statements (cin and cout)
  2) Learn how to process the input into output

Introduction: Input Output

- C++ treats input and output as a **stream** of characters.
- The keywords for input and output operations are stored in the standard library called `iostream`:

```
#include <iostream>
using namespace std;
```

- **cin** → default input stream
- **cout** → default output stream
- Use **`cin`** with **>>** , known as **extraction operator**
- Use **`cout`** with **<<**, known as **insertion operator**

Output: cout

- **cout** command is used to indicate an output stream that will be displayed on the screen (output device)
- The insertion operator takes 2 operands (cout and "what to be displayed")
- Operand on the left is a stream expression (i.e. **cout**).
- Operand on the right is an **expression** or a **string constant**.

Output Statements

SYNTAX

```
cout  <<  Expression    <<  Expression  . . . ;
```

cout statements can be linked together using **<<** operator.

Examples below will produce same output:

```
cout  <<  "Three multiply by five is " ;
cout  <<  3 * 5 ;
```

```
cout  <<"Three multiply by five is "<< 3 * 5 ;
```

Output Statements: String constants

- String constants (in double quotes) are to be printed as is, without the quotes:

```
cout<<"Please enter the number of books ";
```

OUTPUT: Please enter the number of books

- **"Please enter the number of books"** is called a **prompt**.
- All user inputs must be preceded by a **prompt** to tell the user what is expected.
- You must insert **spaces** inside the quotes if you want them in the output.

Output Statements: Expression

- All expressions are computed and then outputted.
- Example 1:

```
cout  <<  "The answer is "  <<  3 * 4 ;
```

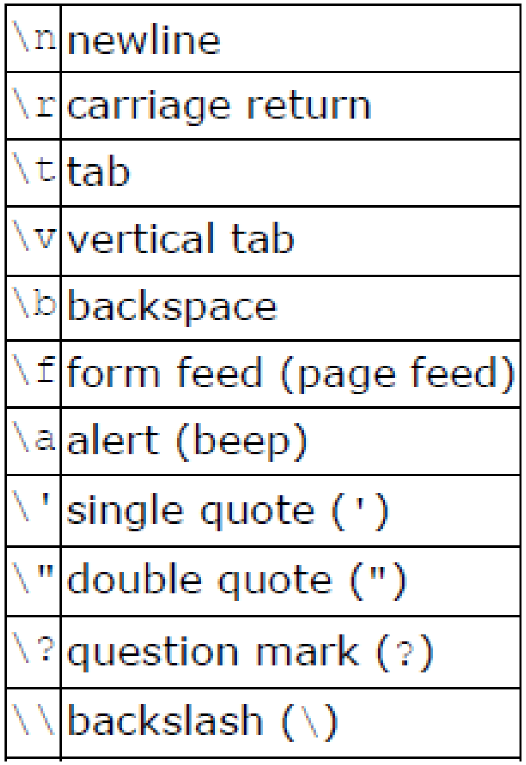
OUTPUT: The answer is 12

- Example 2:

```
int x = 10, y = 12, z;
z = x * y;
cout  <<  "The answer is "  <<  z;
```

OUTPUT: The answer is 120

Output: Escape Sequences



`\n`	newline
`\r`	carriage return
`\t`	tab
`\v`	vertical tab
`\b`	backspace
`\f`	form feed (page feed)
`\a`	alert (beep)
`\'`	single quote (')
`\"`	double quote (")
`\?`	question mark (?)
`\\`	backslash (\)

- In C++, there are techniques can be applied in order to place cursor before or any characters, using an **escape sequence** (a backslash \ followed by a character)

- Example:

```
cout << "\nhai!";//go to new line, and print hai!
cout << "hai!\n";//print hai!, then new line
cout << "ha\ni";//print ha, then i! on new line
```

Output: Newline

- `cout<<"\n"` and `cout<<endl` both are used to insert a blank line.
- Advances the cursor to the start of the next line rather than to the next space.
- Always end the output of all programs with this statement.
- If you there is no *endl* or *\n*, all output will displayed on the same line

Example: Escape Seq.

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
   cout << “1. ” << “Hello there!\n”;
   cout << “2. ” << “Hello\t there!” << endl;
   cout << “3. ” << “Hello\n there!\n”;
   cout << “4. ” << “Hello\t\t there!\n”;
   cout << “5. ” << “Hello there!\a\a\a\a”;
   return 0;
}
```

Output: Formatted Numeric

- Allows the user to control output attributes such as:

  → field width (`setw`, the number of display column that the number takes place)

  → decimal point precision (`setprecision`, number of decimal places

  → number of significant figures
- Must include `#include <iomanip>`

Output: Formatted Numeric (cont.)

- `setw (size)`
  - → sets a MINIMUM width
  - → size must be an integer value
  - → number of significant figures
- Example:

```
int num1 = 1234;
int num2 = 56789;
cout << setw(6) << num1;  // _ _ 1 2 3 4
cout << setw(6) << num1 << setw(6) << num2;  //_ _ 1 2 3 4 _ 5 6 7 8 9
```

Output: Formatted Numeric (cont.)

- `setprecision (num)`
  - → num must be an integer value
  - → the value is rounded up when it is displayed
  - → the precision stays set until it is changed
- Example:

```
double val = 123.456;
cout << setprecision(5) << val; //123.46
```

Output: Formatted Numeric (cont.)

- Floating point format:

  → fixed: print with a fixed number of digits after the decimal point

  → scientific: print in scientific notation

- Example:

```
double y = 50.0512;
cout << fixed << setprecision(2) << y; //50.05
cout << scientific << setprecision(2) << y; //50.05
```

Input: cin

- **cin** command is used to indicate an input stream from the keyboard (input device)
- The extraction operator **>>** takes 2 operands (cin and “where to be stored”)
- Operand on the left is a stream expression (i.e. **cin**).
- Operand on the right is a **variable**

Input: Statements

SYNTAX

```
cin >> Variable  >> Variable . . . ;
```

cin statements can be linked together using **>>** operator.

Examples below will produce same output:

```
cin >> x;
cin >> y;
```

```
cin >> x >> y;
```

How `cin` works?

- Input is not entered until user presses <ENTER> key.
- Allows backspacing to correct.
- Skips whitespaces (space, tabs, etc.)
- Multiple inputs are stored in the order entered:

```
cin >> num1 >> num2;
```

User inputs: **3 4**

Assigns num1 = 3 and num2 = 4

How `cin` works? (cont.)

- Leading blanks for numbers are ignored.
- If the type is double, it will convert integer to double.
- Keeps reading until blank or <ENTER>.
- Remember to **prompt** for inputs

Input: *getline*

- The *getline* function allows us to input characters into a string object
- We can read whole lines of input using:

```
getline (cin, string_name);
```

- Example

```
string username;
cout << "Please key in your name: ";
getline (cin, username);
```

Exercise #1: Input Output

Write a program based on the following pseudocode:

```
print “Exercise for ‘cin’ and ‘cout’”
print “Enter an integer number:”
read int_number
print “Enter a floating point number:”
read float_number
print “Enter a character:”
read aChar
print “Enter double number:”
read double_number
print int_number, float_number, aChar, double_number
```

Exercise #2: Input Output

Get 3 integer numbers from user, and calculate the average!