Unit 3 Lesson 6
Input and Output Streams

with modifications by

Mr. Dave Clausen

Using Cin and Cout

- You must use: #include <iostream.h>
- The << and >> symbols are operators, just as + and * are operators.
- The << symbol is the ***output operator***, and the >> is the ***input operator***.
- The variable or expression to the right of the << or >> operator is what is being input or output.
- cout and cin are actually objects.
- The cout object is the ***destination*** of the output, and the cin object is the ***source*** of the input.
- The >> operator is also referred to as the ***extraction operator***. The << operator is also referred to as the ***insertion operator***.

```
cout << "The number is: " << number;
 cin >> score;
```

Input and Output

- The “arrows” indicate the flow of data in the stream.
- cin (pronounced see-in)
  - Gets data from keyboard, the standard input stream
  - ***extractor operator*** >>
    - Input **from** the standard **input** stream and directed **to a variable** (extract from stream to variable)
- cout (pronounced see-out)
  - ***inserter operator*** <<
    - Insert data **into** standard **output** stream from the literal or expression to the right of the << operator.
  - EGG ILL (to help you remember)
    - Extractor Greater Greater, Inserter Less Less

Streams

- The cin and cout objects are known as ***streams***.
- In C++, a stream is data flowing from one place to another.
- The cin stream reads from what is called the standard input device (keyboard).
- The cout stream leads to the standard output device (monitor).
- Think of streams as “channels” to get to and from devices.

Using Console I/O

- The term console I/O, refers to using the monitor screen and keyboard for input and output.
- The standard use of cin and cout is for console I/O.
- Remember that << is an operator and you may use it multiple times in the same expression, for example:

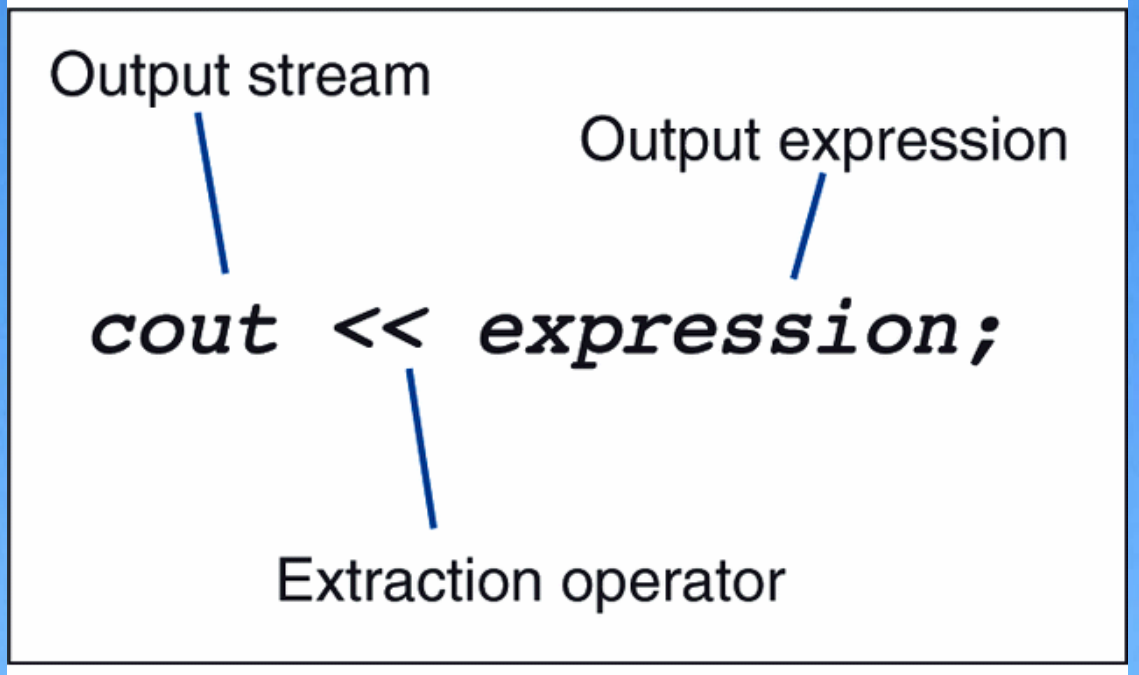
```
cout << “The answer is: “ << answer << endl;
```

Extractor Operator

- Figure 6-1 illustrates the general form of the << operator.

FIGURE 6-1

The << operator is used for output.

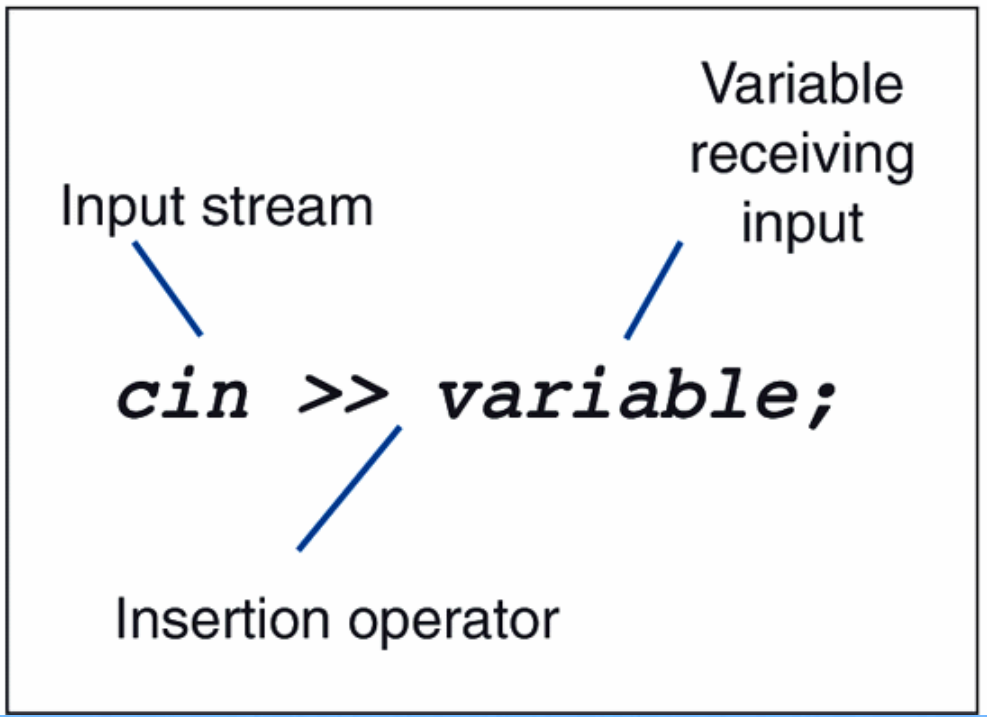


Inserter Operator

- Figure 6-2 illustrates the general form of the >> operator.

FIGURE 6-2

The >> operator is used for input.



Code List 6-1

- Your output can consist of a string literal, a variable, a mathematical expression, or a combination of these as illustrated below in code list 6-1

```
cout << “This string literal will appears on the screen. \n”;
cout << distance;
cout << length_of_room * width_of_room;
cout << “the room is “ << area << “ square feet.\n”;
```

Special Characters

- You have been including ‘\n’ in output statements without a good explanation of what ‘\n’ does.
- It is the new line character or the end of line character.
- It is an important part of formatting output because it causes the cursor to return to the next line of the screen.
- The \n character is one of the special characters available in C++.
- The backslash (\) tells the compiler that a special character is being formed.

What is \n?

- The \n character is called the new line character or the end-of-line character.
- You can use it in any output statement that completes a line.
- The new line character has the same effect in an output statement as pressing the Enter key in a word processor.

Code List 6-2

- The new line character must appear in double quotation marks if it is used in conjunction with other characters, or with a string.
- It may be used with single quotation marks if it appears alone.

```
cout << i << '\n';
// single quotes because it is a single character
cout << "string\n";
// double quotes because it is part of a string
```

Special Characters

- The end-of-line character is called a special character.
- Although \n appears as two characters in the source code, the compiler interprets it as one character.
- The backslash tells the compiler that a special character is being formed.

Table of Special Characters

- Table 6-1 shows other special characters available for use in output statements.

TABLE 6-1
Special characters used in output statements

CHARACTER SEQUENCE	RESULT
\t	Generates a tab character to move the cursor to the next tab stop.
\\	Prints a backslash (\).
\'	Prints a single quotation mark (').
\"	Prints a double quotation mark (").

Using endl

- There is an alternative to \n that you may find easier to enter and more readable.
- You can enter **endl** (end line) in place of '\n', but do not use it as part of a larger string.
- To use endl with string literals, use a statement similar to the following one.
- endl should follow the inserter operator as shown below:

cout << "How now brown cow." << endl;

endl or ‘\n’

- The following commands are functionally identical.
- Because of ease of typing, your instructor, Mr. Clausen prefers endl to ‘\n’.

Code List 6-3

```
cout << i << ‘\n’;
cout << i << endl;
```

Code List 6-4

```
// specchar.cpp                    specchar.txt
// Example of new line and special characters

#include <iostream.h>
int main( )
{
      int i;
      i=25;
      cout << i << '\n';     // single quotes because it is a single character
      cout << "string\n";   // double quotes because it is part of a string
      cout << "The numbers on the following line are separated by tabs.\n";
      cout << "1 \t  2  \t  3  \t  4  \n";
      //  The following lines use end1
      cout << "In C++, you can output backslashes (\\)" << end1;
      cout << "You can also print single quotes (\') and" << end1;
      cout << "double quotes  (\")."  << end1;
      return 0;
}
```

Using setf and unsetf

- The cout object has format options that can be changed.
- To change these options, you send a message to the object using setf and unsetf. (set format & unset format)
- You can left or right justify our output, show the decimal point, or show a fixed number of decimal places for a few of the options.

cout format options

Table 6-2 list the options that can be used.

Option	Description
left	Left-justifies the output
right	Right-justifies the output
showpoint	Displays decimal point and trailing zeros for all floating-point numbers, even if the decimal places are not needed.
uppercase	Displays the “e” in E-notation as “E” instead of “e”
showpos	Displays a leading plus sign for positive values
scientific	Displays floating-point numbers in scientific notation
fixed	Displays floating-point numbers in normal notation

Code List 6-5

```
/ / coutsetf.cpp          coutsetf.txt

#include  <iostream.h>

int  main()
{
     float  x  =  24.0;

     cout  <<  x  <<  end1;                    / /  displays  24

     cout.setf(ios::showpoint);
     cout  <<  x  <<  end1;                   / /  displays  24.0000

     cout.setf(ios::showpos) ;
     cout  <<  x  <<  end1;                   / /   displays  +24.0000

     cout.setf(ios::scientific);
     cout  <<  x  <<  end1;                   / /   displays  +2.400000e+001
```

Code List 6-5 (continued)

```
cout.setf(ios::uppercase);
cout  <<  x  <<  end1;                    / /  displays  +2.400000E+001

cout.unsetf(ios::showpoint);
cout  <<  x  <<  end1;                    / /  displays  +2.400000E+001

cout.unsetf(ios::showpos);
cout  <<  x  <<  end1;                    / /  displays  2.400000E+001

cout.unsetf(ios::uppercase);
cout  <<  x  <<  end1;                    / /  displays  2.400000e+001

cout.unsetf(ios::scientific);
cout << x << endl;                        // displays 24

return  0;
}
```

Using the I/O Manipulators

- Another set of format options is available in C++: the **I/O manipulators.**
- **Don't forget that you must use:**
  #include <iomanip.h>
- **Using setprecision:**
  - When used in conjunction with the fixed format option, the setprecision I/O manipulator sets the number of digits that are to appear to the right of the decimal point.
  - This is very convenient when printing dollars and cents, or any other numbers that require a certain number of decimal places.

Code list 6-7

```
/ /  iomanip.cpp          iomanip.txt

#include  <iostream.h>
#include  <iomanip.h>

int main ()
{
      double cost  =  24.99;
      double total;

      total  =  cost  +  (cost  *  0.07875);  / /  add tax to cost to get total

      / / Display total without formatting
      cout  <<  “Total with no formatting;\n’;
      cout  <<  total  <<  “\n\n”;  / /  use two new line characters to a skip line

      / /  Display total with fixed precision
      cout  <<  “Total with formatting:\n”;
      cout.setf(ios::fixed);
      cout  <<  setprecision(2)  <<  total  <<  “\n\n”;

      return 0;
}
```

setprecision Notes

- Precision is set and will remain until the programmer specifies a new precision with another setprecision command.
  - The decimal uses one position
  - Trailing zeros are printed the specified number of places
  - Leading plus signs are omitted
  - Leading minus signs are printed and use 1 position
  - Digits are rounded, not truncated.

Code list 6-8

- To display dollars and cents refer to the example below:

```
cout << setprecision (2) << '$'
<< total << "\n\n";

// If you need to wrap a long cout statement
   to the next line, press enter before the
   << operator (inserter).
```

Using setw

- The **setw** (set width) manipulator can be used to change the number of spaces the complier uses when it displays a number.
- The amount of space used to display a number is called the ***field width***.
- You can used the setw to set a minimum field width, or use it to format numbers.

Using setw

- The following example allows a field width of 10 for the variables i, j, and k.

```
cout << setw (10) << i << setw (10)
<< j << setw (10) << k << end1;
```

- This is what the output would look like (let * represent a space):

254*******44*********6*********

Code list 6-9 & 6-10

Add the following lines of code to the program iomanip.cpp

//iomanip_final.cpp iomanip_final.txt

Code List 6-9

```
int  i  =  1499;
int  j  =  618;
int  k  =  2;
```

Code List 6-10

```
/ /  Output with no field widths
cout  <<  “Output of i, j, and k with no field widths specified:\n”;
cout  <<  i  <<  j  <<  k  <<  “\n\n”;

/ /  Output with field widths set
cout  <<  “Output of i, j, and k with field widths specified:\n”;
cout  <<  setw (10)  <<  i  <<  setw (10)  <<  j  <<  setw (10)  <<  k
          <<  “\n\n”;
```

Code list 6-11

- The >> operator can be used to input characters. If the user enters more than one character, only the first character will be stored in the variable.

```
/ /  inchar.cpp          inchar.txt
#include  <iostream.h>
int main ()
{
     char c;
     cout  <<  “Enter a single character:  “;
     cin  >>  c;
     cout  <<  “You entered  “  <<  c  <<  ‘\n’;
     return 0 ;
}
```

Inputting Strings

- In lesson 5, you assigned strings to string objects the time of declaration and later, using the assignment operator
- You may have been thinking, “How do I enter a string that is longer than just one word provided by the user into a string object?”
- cin >> will only get characters until you type a white space character (space, tab, enter or newline character).
- As you might have guessed, the string object has a method for doing just that.
- The method is called getline.

Code list 6-12

// instring.cpp instring.txt

```
#inlcude  <iostream.h>
#include  “oostring.h”  //”oostring.cpp for Borland to avoid creating a project.

int main ()
{
      oostring  FirstName;
      oostring  LastName;

      cout  <<  “Enter you first name:  “;
      getline (cin,  FirstName) ;

      cout  <<  “Enter your last name:  “;
      getling (cin,  LastName);

      cout  <<  “Your name us  “  <<  FirstName  <<  “  “  <<  LastName  <<  “\n”;

      return 0 ;
}
```

Flushing the Input Stream

- The cin object is often referred to as the **input stream.**
- Think of the input stream as a line at a checkout stand. Characters are lined up in the input stream as keys are pressed on the keyboard.
- Each character, however, must wait its turn to be processed.
- Therefore, if a prompt for input does not use every character in the input stream, the remaining characters wait in the input stream for the next prompt.

More Input Stream Flushing

- Because characters are left in the input stream, this causes problems when you use a getline after using cin.
- The problems take care of themselves when you are only using cin statements as they “ignore” white space characters (space, tab, enter or newline).

String Input with >>

- The >> operator ignores leading white space
  - space, tab, or enter, new line character, or carriage return
- Then reads nonblank characters
  - until next white space character
  - user is allowed to use backspace or delete
    - until next white space character
- Upon return or white space, string is stored
- >> Can't be used for strings with spaces.

P88Ex1.cpp P88Ex1.txt

String Input with getline

- getline function
  - reads characters, tab, space into string variable
  - until newline (‘/n’) char
  - the newline char is not stored in the string variable
  - getline (<input stream>, <string variable>);
    - getline (cin, name);
  - doesn’t ignore leading white space characters

P88Ex2.cpp P88Ex2.txt

Using cin>> before getline

- >> reads and stores up to newline
- getline reads newline as first char of line
- and quits reading at newline
- any string variables in getline are empty

P89Ex3.cpp P89Ex3.txt

Solutions for >> before getline

- If possible, use getline before cin >>
  (but don't "wimp out" and do this)

  P89Ex4.cpp P89Ex4.txt

- Use:

  getline (cin, consume_newline);
  to consume the newline character. (Preferred method in some textbooks)

  P89Ex5.cpp P89Ex5.txt

- Use:

  cin.get (consume_newline);
  to consume newline character

  P89Ex6.cpp P89Ex6.txt

Our Textbook's Method

- Our textbook prefers the following method of flushing the input stream:

  cin.ignore(80, '\n');
- This tells the input stream to ignore the first 80 characters or the newline character, whichever comes first.
- Enter the command above, before the getline command to flush the input stream.

P89Ex7.cpp P89Ex7.txt

Code list 6-13

```
/ /  flush.cpp                          flush.txt
#include  <iostream.h>
#include  “oostring.h”
int main ()
{
      int  quantity
      oostring  description;
      cout  <<  “Enter the quantity desired:  “;
      cin  >>  quantity;
      cin.ignore( 80,  ‘\n’);  / /  Flush the input stream
      cout  <<  “Enter the description of the item  “;
      getline (cin, description);
      cout  <<  “You requested  “  <<  quantity  <<  “  of item described as  \n”;
      cout  << description <<  “.\n”;
      return  0;
}
```

Using Descriptive Prompts

- When writing programs that interact with the user, be sure to use output prompts that clearly explain the input the program is requesting.
- For example, if prompting the user for his or her name, use a descriptive prompt like the one below.

Please enter your last name:

- If prompting for a telephone number or some other formatted data, you may want to use the prompt to give an example.

Please enter your phone number using the format (555) 555-5555:

- The more descriptive and clear your prompts are, the more likely the user is to enter the information in the form your program is expecting.

Clearing the Screen

- Now that you have learned more about screen I/O, you may be interested in learning how to clear the screen.
- Our compiler has a function available for clearing the screen, called “clear screen”.
- Use #include <conio.h>
- The command is:
- clrscr();
- You should always use a getch(); command on the line before clrscr(); otherwise you won’t be able to see your output. It will disappear before you can read it.