Input Processing – Using the cin Object

CONCEPT: cin along with the stream extraction operator >> can be used to read data typed at the keyboard.
Gathering input from the user is normally a three-step process:
1. Use **cout** to display a prompt on the screen.
2. Use **cin** along with the stream **extraction operator >>** to read a value from the keyboard.
3. Validate the user input prior to processing input data (covered later in Chapter 5 & 6)

The concept of a stream is fundamental to input and output in C++. Similar to how we have described an output stream, you can think of an input stream as a doorway through which characters come into your program from an input device.

As we saw in Chapters 2-3, to use stream input/output (I/O), we write the preprocessor directive #include <iostream>

The header file iostream contains, among other things, the definitions of two data types: istream and ostream. These data types represent input streams and output streams, respectively. The header file also contains declarations that look like this:
istream cin;
ostream cout;

The first declaration says that cin (pronounced "see-in") is a variable of type istream. The second says that our old friend cout is a variable of type ostream. The stream cin is associated with the standard input device (the keyboard).

As you have already seen, you can output values to cout by using the insertion operator (<<), which is sometimes pronounced "put to":
cout << 3 * price;

In a similar fashion, you can input data from cin by using the extraction operator (>>), sometimes pronounced "get from": cin >> cost;
When the computer executes this statement, it inputs the next number you type on the keyboard (425, for ex.) and stores it in the variable cost. The extraction operator >> takes two operands. Its left-hand operand is a stream expression (in the simplest case, just the variable cin). Its right-hand operand is a variable into which we store the input data. For now, let's assume the variable is of a simple type (char, int, float, and so forth). Later in the chapter we discuss the input of string data.

You can use the >> operator several times in a single input statement. Each occurrence extracts (inputs) the next data item from the input stream. For example, there is no difference between the statement
cin >> length >> width;
and the pair of statements
cin >> length;
cin >> width;
Using a sequence of extractions in one statement is very convenient for the programmer. When the user enters characters from the keyboard, they are temporarily placed in an area of memory called the input buffer, or keyboard buffer. cin automatically converts this data to the data type of the variable it is to be stored in. If the user types 10, it is read as the characters '1' and '0', but cin is smart enough to know this will have to be converted to the int value 10 before it is stored in length. If the user enters a floating-point number like 10.7, however, there is a problem. cin knows such a value cannot be stored in an integer variable, so it stops reading when it gets to the decimal point, leaving the decimal point and the rest of the digits in the input buffer. This can cause a problem when the next value is read in. On the same note although it is possible to use cin with the extraction operator >> to input strings, it can also cause problems you need to be aware of. When cin reads data it passes over and ignores any leading whitespace characters (spaces, tabs, or line breaks). However, once it comes to the first nonblank character and starts reading, it stops when it gets to the next whitespace character. Here are a few sample input statements related to the cin object that will be helpful to understand when working with various types of data or stream input

Input Statement

cin>>inputStr; - Skips leading whitespace and stops reading when a trailing whitespace character is encountered (which is not consumed).
cin.get(someChar) – The effect of this function call is to input the next character waiting in the stream- even if it is a whitespace character like a blank line and store it in the variable someChar. The argument to the get function must be a variable, not a constant or arbitrary expression; we must tell the function where we want it to store the input character.
cin.ignore(200, '\n') – The ignore function is used to skip (read and discard) characters in the input stream. The first argument is an int expression; the second, a char value. This function call tells the computer to skip 200 input characters until a newline character is read, whichever comes first.
getline(cin, inputStr); - Does not skip leading whitespace and stops reading when \n is encountered (which is consumed)
cin.clear(): The cin.clear() clears the error flag on cin (so that future I/O operations will work correctly), and then cin.ignore(200, '\n') skips to the next newline (to ignore anything else on the same line as the non-number so that it does not cause another parse failure).
cin.peek(); Returns the next character in the input sequence, without extracting it. The character is left as the next character to be extracted from the stream.

Source code samples featured: GetExample.cpp, GetLineExample.cpp, SimpleIO.cpp, kb_input.cpp, prompts.cpp, charread.cpp, charread2.cpp
cin1.cpp – cin5.cpp – cin processing, cString1.cpp – Cstring5.cpp – cin processing into CStrings
For testing the use of the ignore function during inputs from file data see…
source code: FOURVALS_w_ignore.CPP with the following data files used for testing purposes data files: FOURVALS.dat & FOUR2VAL.dat

---

```
// Program keyboard_input demonstrates various types of input statements may be used to test copy sections into separate programs
#include <iostream>
#include <string>
using namespace std;

int main()
{
   cout  << fixed  << showpoint;
   int val1, val2, val3, val4;           // declares 4 variables
                          //missing statement ??
   cin >> val1 >> val2 >> val3 >> val4;            // inputs 4 values
   cout  << val4  << endl;
   cout  << val3  << endl;
   cout  << val2  << endl;
   cout  << val1  << endl;              // outputs 4 values

   char  character1, character2, character3, character4;
   cout  << "Input four characters.  Press Return."  << endl;
   cin  >> character1  >> character2  >> character3  >> character4;
   cout  << character1  << character2  << character3  << character4<<endl;

   //cin.get
   cout  << "Input four characters again.  Press Return."  << endl;
   cin.get(character1);
   cin.get(character2);
   cin.get(character3);
   cin.get(character4);
   cout  << character1  << character2  << character3  << character4<<endl;

  //mixing cin>> and cin.get
  char somecharacter;    //define a character variable
  int number;          //define an integer variable
  cout<<"Enter a number: ";
  cin>>number;          //read an integer
  cout<<"Enter character: ";
  somecharacter=cin.get(); //read a character
  cout<<"Thank You!\n\n";

  //using cin.ignore  ....forms cin.ignore(), cin.ignore(20,'\n') ...cin.ignore(n,c)
   cout<<"Enter a number: ";
  cin>>number;          //read an integer
  cin.ignore();          //Skip the newline character
  cout<<"Enter character: ";
  cin.get(somecharacter) ; //read a character
  cout<<"Thank You!\n";

  //getline
   string name, city;

   cout << "Please enter your name: ";
   getline(cin, name);
   cout << "Enter the city you live in: ";
   getline(cin, city);

   cout << "Hello, " << name << endl;
   cout << "You live in " << city << endl;
   return 0;
}
```