**Chapter 13: Stream Input/Output: A Deeper Look**

**Section 13.1 Introduction**

13.1 Q1: Which of the following statements is *not* true about C++ input/output (I/O) features?

1. C++ allows users to specify I/O for their own data types.
2. C++ automatically calls the correct I/O operations for standard types.
3. C++ has some object-oriented I/O features.
4. C++ borrowed its type safe I/O capabilities from C.

**ANS d. C++ borrowed its type safe I/O capabilities from C.**

**Section 13.2 Streams**

13.2 Q1: Select the *correct* statement regarding C++ I/O streams:

1. C++ provides only high-level I/O capabilities because it is a high-level programming language.
2. High-level (formatted) I/O is best for large-volume transfers.
3. Low-level I/O breaks information down into small, meaningful groups of related bytes.
4. Programmers generally prefer high-level I/O to low-level I/O.

**ANS d. Programmers generally prefer high-level I/O to low-level I/O.**

13.2 Q2: \_\_\_\_\_\_\_\_\_\_ is usually *faster* than \_\_\_\_\_\_\_\_\_\_.

1. High-level I/O, low-level I/O.
2. Low-level I/O, high-level I/O.
3. Low-level I/O, internal data processing.
4. High-level I/O, internal data processing.

**ANS b. Low-level I/O, high-level I/O.**

**Section 13.2.1 Classic Streams vs. Standard Streams**

13.2.1 Q1: Which C++ data type was designed to store Unicode characters?

1. char
2. long
3. wchar\_t
4. size\_t

**ANS c. wchar\_t**

13.2.1 Q2 [C++11] Which of the following statements is false?

1. The C++ standard redesigned the classic C++ stream classes, which processed only chars, as class templates with specializations for processing characters of types char and wchar\_t, respectively.
2. C++11 added type char64\_t to handle the new double-width Unicode characters.
3. The size of type wchar\_t is not specified by the C++ standard.
4. C++11’s new char16\_t and char32\_t types for representing Unicode characters were added to provide character types with explicitly specified sizes.

ANS: b. C++11 added type char64\_t to handle the new double-width Unicode characters.

**Section 13.2.2 iostream Library Header Files**

No questions.

**Section 13.2.3 Stream Input/Output Classes and Objects**

13.2.3 Q1: Which of the following classes is a *base class* of the other three?

1. basic\_ios
2. basic\_istream
3. basic\_ostream
4. basic\_iostream

**ANS a. basic\_ios**

13.2.3 Q2: Which of the following is *not* an object of the ostream class?

1. cout
2. cerr
3. cin
4. clog

**ANS c. cin**

13.2.3 Q3: Which of the following classes is *deepest* in the inheritance hierarchy?

1. basic\_iostream
2. basic\_ofstream
3. basic\_ifstream
4. basic\_fstream

**ANS d. basic\_fstream**

**Section 13.3 Stream Output**

13.3 Q1: Which of the following is *not* a member function of the C++ ostream class?

1. Stream-insertion operator (<<).
2. Stream-extraction operator (>>).
3. put.
4. write.

**ANS b. Stream-extraction operator (>>).**

**Section 13.3.1 Output of char \* Variables**

13.3.1 Q1: Which of the following prints the address of character string string given the following declaration?

char \* string = "test";

1. cout << string;
2. cout << \*&string;
3. cout << static\_cast< void \* >( string );
4. cout << \* string;

**ANS: c. cout << static\_cast< void \* >( string );**

**Section 13.3.2 Character Output using Member Function put**

13.3.2 Q1: Which of the following is an *illegal* use of function put?

1. cout.put( 'A' );
2. cout.put( "A" );
3. cout.put( 'A' ).put( '\n' );
4. cout.put( 65 );

**ANS: b. cout.put( "A" );**

**Section 13.4 Stream Input**

13.4 Q1: The stream-extraction operator:

1. Does *not* normally accept white-space characters.
2. Returns true when the end-of-file is encountered.
3. Sets the stream’s failbit if the operation fails.
4. Sets the stream’s badbit if the data is of the wrong type.

**ANS: a. Does *not* normally accept white-space characters.**

**Section 13.4.1 get and getline Member Functions**

13.4.1 Q1: One *difference* between the three-argument version of the get function and the getline function is that:

1. Only get has a delimiter.
2. The getline function removes the delimiter from the stream.
3. Only get adds the delimiter to the array.
4. getline stores the characters it reads into its character array argument.

**ANS: b. The getline function removes the delimiter from the stream.**

**Section 13.4.2 istream Member Functions peek, putback and ignore**

13.4.2 Q1: The putback member function returns to the input stream the previous character obtained by:

1. A get from the input stream.
2. Using the stream extraction operator on the input stream.
3. Reading input from the keyboard.
4. Reading a file from disk.

**ANS: a. A get from the input stream.**

13.4.2 Q2: Upon encountering the designated delimiter character, the ignore member function will:

1. Read it in and return its value.
2. Ignore it and continue reading and discarding characters.
3. Terminate.
4. Replace it with an EOF character.

**ANS: c. Terminate.**

**Section 13.4.3 Type-Safe I/O**

13.4.3 Q1: If *unexpected* data is processed in an I/O operation:

1. An exception will be thrown.
2. An error message will automatically be displayed.
3. The program will terminate execution.
4. Various error bits will be set.

**ANS: d. Various error bits will be set.**

**Section 13.5 Unformatted I/O using read, write and gcount**

13.5 Q1: Which of the following is a *difference* between the read and write functions?

1. One performs formatted I/O and the other does not.
2. They take *different* types of parameters.
3. write and gcount are member functions of the same class, whereas read is not.
4. The failbit is set *only* with read.

**ANS: d. The failbit is set *only* with read.**

**Section 13.6 Introduction to Stream Manipulators**

**Section 13.6.1 Integral Stream Base: dec, oct, hex and setbase**

13.6.1 Q1: Which of the following is *not* a difference between hex and setbase?

1. setbase is a parameterized stream manipulator and hex is not.
2. setbase is provided by a *different* header file than hex.
3. setbase(16) and hex have *different* effects on stream output.
4. setbase takes an argument but hex does not.

**ANS: c. setbase(16) and hex have *different* effects on stream output.**

**Section 13.6.2 Floating-Point Precision (precision, setprecision)**

13.6.2 Q1: What will be *output* by the following statements?

double x = 1.23456789;  
cout << fixed;

cout << setprecision(5) << x << endl;

cout.precision(3);

cout << x << endl;

cout << x << endl;

1. 1.2346  
   1.23  
   1.23
2. 1.23457  
   1.235  
   1.23456789
3. 1.2346  
   1.23  
   1.23456789
4. 1.23457  
   1.235  
   1.235

**ANS: d. 1.23457**

**1.235**

**1.235**

**Section 13.6.3 Field Width (width, setw)**

13.6.3 Q1: Which of the following is *not* true about setw and width?

1. If the width set is *not* sufficient the output prints as wide as it needs.
2. They are used to set the field width of output.
3. Both of them can perform two tasks, setting the field width and returning the current field width.
4. They only apply for the next insertion/extraction.

**ANS: c. Both of them can perform two tasks, setting the field width and returning the current field width.**

**Section 13.6.4 User-Defined Output Stream Manipulators**

13.6.4 Q1: Which of the following is a *valid* user-defined output stream manipulator header?

1. ostream& tab( ostream& output )
2. ostream tab( ostream output )
3. istream& tab( istream output )
4. void tab( ostream& output )

**ANS: a. ostream& tab( ostream& output )**

**Section 13.7 Stream Format States and Stream Manipulators**

**Section 13.7.1 Trailing Zeros and Decimal Points (showpoint)**

**[\*\*\*P: Confirm in code\*\*\*]** 13.7.1 Q1: What will be *output* by the following statement?

cout << showpoint << setprecision(4) << 11.0 << endl;

1. 11
2. 11.0
3. 11.00
4. 11.000

**ANS: c. 11.00**

**Section 13.7.2 Justification (left, right and internal)**

13.7.2 Q1: Which of the following stream manipulators causes an outputted number’s sign to be *left justified*, its magnitude to be *right justified* and the center space to be filled with fill characters?

1. left
2. right
3. internal
4. showpos

**ANS: c. internal**

**Section 13.7.3 Padding (fill, setfill)**

13.7.3 Q1: Which of the following statements restores the *default fill character*?

1. cout.defaultFill();
2. cout.fill();
3. cout.fill( 0 );
4. cout.fill( ' ' );

**ANS: d. cout.fill( ' ' );**

**Section 13.7.4 Integral Stream Base (dec, oct, hex, showbase)**

13.7.4 Q1: When the showbase flag is set:

1. The base of a number precedes it in brackets.
2. Decimal numbers are *not* output any differently.
3. "oct" or "hex" will be displayed in the output stream.
4. Octal numbers can appear in one of two ways.

**ANS: b. Decimal numbers are not output any differently.**

**Section 13.7.5 Floating-Point Numbers; Scientific and Fixed Notation (scientific, fixed)**

13.7.5 Q1: What will be *output* by the following statements?

double x = .0012345;  
 cout << fixed << x << endl;

cout << scientific << x << endl;

1. 1.234500e-003  
   0.001235
2. 1.23450e-003  
   0.00123450
3. .001235  
   1.234500e-003
4. 0.00123450  
   1.23450e-003

**ANS: c. 0.001235**

**1.234500e-003**

**Section 13.7.6 Uppercase/Lowercase Control (uppercase)**

13.7.6 Q1: Which of the following outputs does *not* guarantee that the uppercase flag has been set?

1. All hexadecimal numbers appear in the form 0X87.
2. All numbers written in scientific notation appear the form 6.45E+010.
3. All text outputs appear in the form SAMPLE OUTPUT.
4. All hexadecimal numbers appear in the form AF6.

**ANS: c. All text outputs appear in the form SAMPLE OUTPUT.**

**Section 13.7.7 Specifying Boolean Format (boolalpha)**

13.7.7 Q1: Which of the following is *not* true about bool values and how they're output with the output stream?

1. The old style of representing true/false values used -1 to indicate false and 1 to indicate true.
2. A bool value outputs as 0 or 1 by default.
3. Stream manipulator boolalpha sets the output stream to display bool values as the strings "true" and "false".
4. Both boolalpha and noboolalpha are “sticky” settings.

**ANS: a. The old style of representing true/false values used -1 to indicate false and 1 to indicate true.**

**Section 13.7.8 Setting and Resetting the Format State via Member Function flags**

13.7.8 Q1: To reset the format state of the output stream:

1. Call the reset member function.
2. Call the flags member function with the ios\_base::fmtflags constant as the argument.
3. Save a copy of the fmtflags value returned by calling member function flags *before* making any format changes, and then call flags again with that fmtflags value as the argument.
4. You must manually apply each individual format change member function or stream manipulator to restore the default format state.

**ANS: c. Save a copy of the fmtflags value returned by calling member function flags *before* making any format changes, and then call flags again with that fmtflags value as the argument.**

**Section 13.8 Stream Error States**

13.8 Q1: The good member function will return false if:

1. The eof member function would return true.
2. The bad member function would return true.
3. The failbit member function would return true.
4. Any of the above.

**ANS: d. Any of the above.**

13.8 Q2: The difference between the operator! member function and the operator void\* member function is that:

1. They always return opposite boolean values.
2. They occasionally return opposite boolean values.
3. Of the two member functions, only operator! checks if eof has been set.
4. Of the two member functions, only operator void\* checks if eof has been set.

**ANS: a. They always return opposite boolean values.**

**Section 13.9 Tying an Output Stream to an Input Stream**

13.9 Q1: Select the *false* statement. Outputs are:

1. Flushed automatically at the end of a program.
2. Flushed when the buffer fills.
3. Able to be synchronized with inputs.
4. Never automatically tied to inputs.

**ANS: d. Never automatically tied to inputs.**

13.9 Q2: Untying an input stream, inputStream, from an output stream, outputStream, is done with the function call:

1. inputStream.untie().
2. inputStream.untie( &outputStream ).
3. inputStream.tie().
4. inputStream.tie( 0 ).

**ANS: d. inputStream.tie( 0 ).**