1. Use table below to answer simplify the expressions that follow.

C++ Operator Precedence

The following table lists the precedence and associativity of C++ operators. Operators are listed top to bo descending precedence.

Precedence	Operator	Description	Associativity
1	`::`	Scope resolution	Left-to-right
2	`++ --`	Suffix/postfix increment and decrement	
	type`()` *type*`{}`	Function-style type cast	
	`()`	Function call	
	`[]`	Array subscripting	
	`.`	Element selection by reference	
	`->`	Element selection through pointer	
3	`++ --`	Prefix increment and decrement	Right-to-left
	`+ -`	Unary plus and minus	
	`! ~`	Logical NOT and bitwise NOT	
	(*type*)	C-style type cast	
	`*`	Indirection (dereference)	
	`&`	Address-of	
	`sizeof`	Size-of[note 1]	
	`new, new[]`	Dynamic memory allocation	
	`delete, delete[]`	Dynamic memory deallocation	
4	`.* ->*`	Pointer to member	Left-to-right
5	`* / %`	Multiplication, division, and remainder	
6	`+ -`	Addition and subtraction	
7	`<< >>`	Bitwise left shift and right shift	
8	`< <=`	For relational operators < and ≤ respectively	
	`> >=`	For relational operators > and ≥ respectively	
9	`== !=`	For relational = and ≠ respectively	
10	`&`	Bitwise AND	
11	`^`	Bitwise XOR (exclusive or)	
12	`|`	Bitwise OR (inclusive or)	
13	`&&`	Logical AND	
14	`||`	Logical OR	
15	`?:`	Ternary conditional[note 2]	Right-to-left
	`=`	Direct assignment (provided by default for C++ classes)	
	`+= -=`	Assignment by sum and difference	
	`*= /= %=`	Assignment by product, quotient, and remainder	
	`<<= >>=`	Assignment by bitwise left shift and right shift	
	`&= ^= |=`	Assignment by bitwise AND, XOR, and OR	
16	`throw`	Throw operator (for exceptions)	
17	`,`	Comma	Left-to-right

(a) $-4-3*(4*4-4*3*2)/2*3$
ANS: $-4-3*(4*4-4*3*2)/2*3 = -4-3*(16-24)/2*3 = -4-3*(-8)/2*3$
$= -4-(-24)/2*3 = -4-(-12)*3 = -4-(-36) = 32$

(b) $13\%4-1/23\%4$
ANS: $13\%4-1/23\%4 = 1-0\%4 = 1-0 = 1$

Determine whether the following boolean expressions are `true` or `false`. Assume that the value of the integer variable `count` is 0 and the value of the integer variable `limit` is 10.

(c) `(count == 0) && (limit < 20) = true && true =` `true`

(d) `count == 0 && limit < 20` **ANS:** Since comparison operators take precedence over logical "AND", the result here is the same as above.

(e) `(limit < 20) || (count == 5) = true OR false =` `true`

(f) `!(count == 12) = !false =` `true`

(g) `(count == 1) && (x < y) = false AND ? =` `false`

(h) `(count < 10) || (x < y) = true OR ? =` `true`

(i) `!( ((count < 10) || (x < y)) && (count >= 0) ) = !((true OR ?) AND true = !true AND true = false AND true =` `false`

(j) `((limit < 20) || (limit/count) > 7)) = true OR ? =` `true`

(k) `(5 && 7) + (!6) = true AND true + false = true AND true false =` `true`

2. Consider the code below:

```
#include "std_lib_facilities.h"
//------------------------------------------------------------------------------
double some_function() {
    double d = 0;
    cin >> d;
    if (!cin) error("couldn't read a double in 'some_function()'");
    // do something useful
}
//------------------------------------------------------------------------------
int main()
try {
    // our program
    some_function();
    return 0;    // 0 indicates success
}
catch (exception& e) {
    cerr << "error: " << e.what() << '\n';
    keep_window_open();
    return 1;    // 1 indicates failure
}
catch (...) {
    cerr << "Oops: unknown exception!\n";
    keep_window_open();
    return 2;    // 2 indicates failure
}
//------------------------------------------------------------------------------
```

(a) What happens if the user types `1.e-5 return` in the console?
ANS: Nothing.

(b) What happens if the user types `Q return` in the console?
ANS: Since the input is not in the form or a `double` type, `cin` will go into a `fail` state so `!cin` will be `true` resulting in `some_function()` throwing an error with the message "`couldn't read a double in some_function()`"

3. Write a program to to allow a user to enter two different `vector`s of 10 `int`s, and then compute the sum of their component products: `v1[0]*v2[0]+v1[1]*v2[1]+···+v1[9]*v2[9]`

```
#include <vector>
#include <iostream>

int dotProd(std::vector<int>, std::vector<int>);
std::vector<int> getVector(int n);

int main() {
    std::vector<int> v1 = getVector(10); //magic 10
    std::vector<int> v2 = getVector(10);
    std::cout << "The dot product of your vectors is " << dotProd(v1, v2);
}

int dotProd(std::vector<int> v1, std::vector<int> v2) {
    if (v1.size() != v2.size()) return 2147483647; //absurd result
    int sum{ 0 };
    for (int i = 0; i < v1.size(); ++i)
        sum += v1[i] * v2[i];
    return sum;
}

std::vector<int> getVector(int n) {
    int xin;
    std::vector<int> returnV(n);
    std::cout << "Enter " << n << " integers: ";
    for (int i = 0; i < n; ++i) {
        std::cin >> xin;
        returnV[i] = xin;
    }
    return returnV;
}
//-----------------------------------------------------------------------------
```