50 Tricky C++ Programs — Cheatsheet (with Expected Outputs)

Generated: 2025-09-20 08:20:37

Topics: iostream, data types, casting, overflow/underflow, iomanip, operators & precedence, if quirks, math-only tricks, logic puzz

1.
```
// 1. Mixing >> and getline() but with leading whitespace
#include<iostream>
#include<string>
using namespace std;
int main(){
    int n;
    string s;
    cin >> n;        // input: 3<enter><space>hello<enter>
    getline(cin, s);
    cout << "n=" << n << "|s=" << s.size() << ":" << s;
}
```

Expected output: Input: `3\n hello\n` Output: `n=3|s=6: hello` (note leading space included; getline reads remainder including the leading space)

2.
```
// 2. bool output formatting with arithmetic
#include<iostream>
using namespace std;
int main(){
    bool x = 5; cout << x << " " << boolalpha << x;
}
```

Expected output: Output: `1 true`

3.
```
// 3. cin >> char reads skips whitespace; careful with getline vs >> mixing
#include<iostream>
using namespace std;
int main(){
    char c1, c2;
    cin >> c1;      // input: 'a' <space> ' ' <enter>
    c2 = cin.get(); // deliberately use get()
    cout << int(c1) << ' ' << int(c2);
}
```

Expected output: If input `a (space)\n` then output: `97 32` (32 is space ASCII).

4.
```
// 4. Left shift and precedence with cout
#include<iostream>
using namespace std;
int main(){
    int a=5;
    cout << (a<<2) << " " << (a>>1);
}
```

Expected output: Output: `20 2`

5.
```
// 5. endl flush vs '\\n' — but with cout chaining and flush timing
#include<iostream>
using namespace std;
int main(){
    cout << "A" << endl << "B" << '\n' << "C";
}
```

Expected output: Output: `A` then newline `B` then newline `C` (final: `A\nB\nC`).

6.
```
// 6. char overflow with value >127 assigned to signed char
#include<iostream>
using namespace std;
int main(){
    signed char c = 130;
    cout << int(c);
}
```

Expected output: On typical two's complement 8-bit char: 130 -> -126 so output: `-126`.
(Depends on implementation if char is signed.)

7.
```
// 7. sizeof('a') is size of int in C++ (character literal is int), sizeof("a") is
pointer-to-char array size
#include<iostream>
using namespace std;
int main(){
    cout << sizeof('a') << " " << sizeof(\"a\");
}
```

Expected output: Common output on many compilers: `4 2` (character literal is int -> 4
bytes; string literal "a" has 2 chars including '\0').

8.
```
// 8. signed vs unsigned comparison surprising result
#include<iostream>
using namespace std;
int main(){
    int a = -1;
    unsigned int b = 1;
    cout << (a < b);
}
```

Expected output: Output: `0` or `false`? Actually `-1` converts to large unsigned
(UINT_MAX) so (a<b) is false -> `0`.

9.
```
// 9. float vs double precision equality
#include<iostream>
using namespace std;
int main(){
    float x = 0.1f;
    double y = 0.1;
    cout << (x==y);
}
```

Expected output: Output: `0` (false) because float 0.1f != double 0.1 due to precision
differences.

10.
```
//10. char arithmetic prints integer result
#include<iostream>
using namespace std;
int main(){
    char c = 'A';
    cout << c+1;
}
```

Expected output: Output: `66` (because 'A' is 65; arithmetic promotes to int).

11.

```
//11. truncation when casting float to int
#include<iostream>
using namespace std;
int main(){
    float f = 3.9f;
    cout << (int)f;
}
```

Expected output: Output: `3` (truncation toward zero).

12.

```
//12. double to char with fractional part
#include<iostream>
using namespace std;
int main(){
    double d = 97.99;
    cout << char(d) << ' ' << int(d);
}
```

Expected output: Output: `a 97` (char(97.99) becomes 'a' after truncation).

13.

```
//13. integer division vs float division
#include<iostream>
using namespace std;
int main(){
    cout << 5/2 << " " << 5/2.0;
}
```

Expected output: Output: `2 2.5`

14.

```
//14. explicit cast after division
#include<iostream>
using namespace std;
int main(){
    cout << (int)(7/2.0);
}
```

Expected output: 7/2.0 is 3.5 -> cast to int -> `3`

15.

```
//15. unsigned cast negative converts to large positive
#include<iostream>
using namespace std;
int main(){
    cout << (unsigned)-5;
}
```

Expected output: Output: very large number equal to UINT_MAX-4 (e.g., `4294967291` on 32-bit unsigned).

16.

```
//16. int overflow (undefined behaviour) — often wraps on two's complement but UB
#include<iostream>
using namespace std;
int main(){
    int x = 2147483647;
    cout << x+1;
```

```
}
```

Expected output: Undefined behavior. On many compilers with two's complement it wraps to `-2147483648` but do not rely on it.

17.

```
//17. unsigned wrap-around is well-defined
#include<iostream>
using namespace std;
int main(){
    unsigned int x = 0;
    cout << x-1;
}
```

Expected output: Output: `4294967295` on 32-bit unsigned (i.e., UINT_MAX).

18.

```
//18. float overflow to inf
#include<iostream>
using namespace std;
int main(){
    float f = 1e38f;
    cout << f*1000;
}
```

Expected output: Likely output: `inf` (infinity) or very large value; may print `inf`.

19.

```
//19. signed char overflow example
#include<iostream>
using namespace std;
int main(){
    signed char c = 127;
    cout << int(c+1);
}
```

Expected output: If char is signed 8-bit and arithmetic done in int: c+1 -> 128 then printing int -> `128`. But if stored into signed char then overflow UB. So output commonly `128`.

20.

```
//20. mixed signed/unsigned comparison again
#include<iostream>
using namespace std;
int main(){
    unsigned int u = 0;
    int i = -1;
    cout << (i<u);
}
```

Expected output: i converts to unsigned -> large -> (i<u) false -> `0`.

21.

```
//21. setw and setfill combine; fill persists
#include<iostream>
#include<iomanip>
using namespace std;
int main(){
    cout << setw(5) << setfill('*') << 42 << setw(3) << 7;
}
```

Expected output: Output: `**42**7`? Explanation: first prints `**42` (width 5), then

setw(3) applies to 7 -> `  7` but setfill still '*' so becomes `**7`. Full: `**42**7`

22.

```
//22. fixed vs scientific toggles persist
#include<iostream>
#include<iomanip>
using namespace std;
int main(){
    double d = 1234.56789;
    cout << fixed << setprecision(2) << d << " "
         << scientific << setprecision(2) << d;
}
```

Expected output: Output: `1234.57 1.23e+03` (exact formatting may vary slightly like `1.23e+03`).

23.

```
//23. left keeps alignment until changed
#include<iostream>
#include<iomanip>
using namespace std;
int main(){
    cout << left << setw(10) << 42 << "X";
}
```

Expected output: Output: `42        X` (42 then spaces to width 10, then X).

24.

```
//24. boolalpha persists
#include<iostream>
#include<iomanip>
using namespace std;
int main(){
    cout << boolalpha << (5>3) << " " << (0);
}
```

Expected output: Output: `true 0` (boolalpha affects booleans, but printing 0 as int remains `0`).

25.

```
//25. hex with uppercase and showbase
#include<iostream>
#include<iomanip>
using namespace std;
int main(){
    int n=255;
    cout << showbase << uppercase << hex << n;
}
```

Expected output: Output: `0XFF`

26.

```
//26. Undefined order of evaluation with ++ used twice (UB)
#include<iostream>
using namespace std;
int main(){
    int a=5;
    cout << ++a + a++;
}
```

Expected output: Undefined behavior — could print different values like `12` or `11`. Do not rely on it.

27.
```
//27. assignment inside expression evaluated left-to-right? (sequence points changed in
C++17)
#include<iostream>
using namespace std;
int main(){
    int a=5;
    cout << (a=3)+(a=4);
}
```

Expected output: In C++17 evaluation order is well-defined left-to-right for operator+? Still this is unspecified historically. On many compilers result `7` because (a=3) yields 3 then (a=4) yields 4 -> 7.

28.
```
//28. nested ternary with subtle grouping
#include<iostream>
using namespace std;
int main(){
    int a=5;
    cout << (a<5?1:(a==5?2:3));
}
```

Expected output: Output: `2`

29.
```
//29. bitwise & vs logical &&
#include<iostream>
using namespace std;
int main(){
    cout << (3&1) << " " << (3&&1);
}
```

Expected output: Output: `1 1` (bitwise AND gives 1; logical AND of non-zero values is true -> 1).

30.
```
//30. modulo with negative dividend
#include<iostream>
using namespace std;
int main(){
    cout << (-7)%3 << " " << (-7+3*((-7)/3));
}
```

Expected output: On C++ standard, -7%3 == -1 and -7/3 == -2. So output: `-1 -1`

31.
```
//31. if with assignment (common trap)
#include<iostream>
using namespace std;
int main(){
    int a=5, b=10;
    if(a=b) cout<<a;
}
```

Expected output: if (a=b) assigns b to a (a becomes 10), then condition true -> prints `10`

32.
```
//32. chained if with assignment
#include<iostream>
```

```
using namespace std;
int main(){
    int x=0;
    if(x=1) cout<<\"One\"; else cout<<\"Zero\";
}
```

Expected output: Assignment makes x=1 -> condition true -> prints `One`

33.

```
//33. dangling else demonstration
#include<iostream>
using namespace std;
int main(){
    int a=5;
    if(a>0) if(a<10) cout<<\"Y\"; else cout<<\"N\";
}
```

Expected output: Outputs `Y`. The else pairs with inner if.

34.

```
//34. empty if statement with semicolon
#include<iostream>
using namespace std;
int main(){
    if(0); else cout<<\"Else works\";
}
```

Expected output: Output: `Else works` (because if(0); does nothing, else executes).

35.

```
//35. assignment vs comparison common pitfall
#include<iostream>
using namespace std;
int main(){
    int x=0;
    if(x=5) cout<<\"Assign\";
}
```

Expected output: Assigns 5 -> condition true -> prints `Assign`

36.

```
//36. abs, max using math (no if)
#include<iostream>
#include<cmath>
using namespace std;
int main(){
    int a=3,b=7;
    cout << (a+b+abs(a-b))/2 << " " << (a+b-abs(a-b))/2;
}
```

Expected output: Output: `7 3` (max then min)

37.

```
//37. even/odd without if using arithmetic indexing
#include<iostream>
using namespace std;
int main(){
    int x=7;
    const char* t[] = {\"Even\",\"Odd\"};
    cout << t[x%2];
}
```

Expected output: Output: `Odd`

38.

```
//38. sign of number with maths only
#include<iostream>
using namespace std;
int main(){
    int x=-42;
    cout << (x>0)-(x<0);
}
```

Expected output: Output: `-1` (negative)

39.

```
//39. clever swap without temp using XOR (watch aliasing)
#include<iostream>
using namespace std;
int main(){
    int a=5,b=7;
    a ^= b; b ^= a; a ^= b;
    cout << a << ' ' << b;
}
```

Expected output: Output: `7 5` (but unsafe if a and b alias same variable).

40.

```
//40. compute max of three using math
#include<iostream>
#include<algorithm>
using namespace std;
int main(){
    int a=2,b=9,c=5;
    cout << max(a, max(b,c));
}
```

Expected output: Output: `9`

41.

```
//41. leap year tricky
#include<iostream>
using namespace std;
int main(){
    int y=1900;
    if((y%400==0)||((y%4==0)&&(y%100!=0))) cout<<\"Leap\"; else cout<<\"Not Leap\";
}
```

Expected output: Output: `Not Leap` (1900 is not leap).

42.

```
//42. short-circuit avoids division by zero
#include<iostream>
using namespace std;
int main(){
    int a=0;
    if(a && 10/a) cout<<\"Yes\"; else cout<<\"No\";
}
```

Expected output: Output: `No` (second operand not evaluated due to short-circuit).

43.

```
//43. switch fallthrough trap
#include<iostream>
```

```
using namespace std;
int main(){
    int n=2;
    switch(n){
        case 1: cout<<\"One\";
        case 2: cout<<\"Two\";
        default: cout<<\"Other\";
    }
}
```

Expected output: Output: `TwoOther` (fallthrough from case 2 to default).

44.

```
//44. nested ternary concise comparison
#include<iostream>
using namespace std;
int main(){
    int n=0;
    cout << (n>0?\"Pos\":(n<0?\"Neg\":\"Zero\"));
}
```

Expected output: Output: `Zero`

45.

```
//45. if with logical NOT and 0/1 mapping
#include<iostream>
using namespace std;
int main(){
    int n=5;
    if(!(n%2)) cout<<\"Even\"; else cout<<\"Odd\";
}
```

Expected output: Output: `Odd`

46.

```
//46. implicit bool conversion with pointers and integers
#include<iostream>
using namespace std;
int main(){
    int x=10;
    if(x) cout<<\"True\"; else cout<<\"False\";
}
```

Expected output: Output: `True`

47.

```
//47. uninitialized variable used in if assignment (UB)
#include<iostream>
using namespace std;
int main(){
    int a;
    if(a=0) cout<<\"Zero\"; else cout<<\"Else\";
}
```

Expected output: Assigns 0 to a -> prints `Else`. (Be careful: reading uninitialized vars is UB but assignment is fine.)

48.

```
//48. comma operator returns last expression
#include<iostream>
using namespace std;
int main(){
```

```
    int a=1,b=0;
    if(a,b) cout<<\"b\"; else cout<<\"a\";
}
```

Expected output: Comma operator evaluates a then b and returns b -> value 0 -> condition false -> prints `a`

49.

```
//49. bitwise OR in if (nonzero becomes true)
#include<iostream>
using namespace std;
int main(){
    int x=0;
    if(x|1) cout<<\"True\";
}
```

Expected output: x|1 -> 1 -> condition true -> Output: `True`

50.

```
//50. dangling else with two-level ifs
#include<iostream>
using namespace std;
int main(){
    int x=0,y=1;
    if(x) if(y) cout<<\"Y\"; else cout<<\"N\";
}
```

Expected output: No output (neither Y nor N) because outer if false; dangling else pairs with inner if only if inner evaluated.
