

# CS2313 Computer Programming

## LT4 – Conditional Statements



香港城市大學  
City University of Hong Kong

專業 創新 胸懷全球  
Professional · Creative  
For The World

# Outlines

- if statement
  - Simple
  - Nested
- Boolean logic
- switch statement

# Outcomes

- Work out the **boolean value** of a logic expression.
- Express decision making logic in **boolean logic**.
- Using **if** / **switch** statement to express **conditional** instruction.

# Syntax Summary

- **Keywords**
  - `if`, `else`, `switch`, `case`, `default`.
- **Punctuators**
  - `( ... )`
  - `{ ... }`
  - `:`
  - `? :`
- **Operators**
  - `==`, `!`, `!=`, `>`, `>=`, `<`, `<=`, `&&`, `||`

# Comparative Operators

- **Binary operators** which accept two operands and compare them:

Relational operators	Syntax	Example
Less than	<	x<y
Greater than	>	z>1
Less than or equal to	<=	b<=1
Greater than or equal to	>=	c>=2

Equality operators	Syntax	Example
Equal to	==	a==b
Not equal to	!=	b !=3

# Simplified Expression

Original Expression	Simplified Expression
$! (x < y)$	$x \geq y$
$! (x > y)$	$x \leq y$
$! (x \neq y)$	$x = y$
$! (x \leq y)$	$x > y$
$! (x \geq y)$	$x < y$
$! (x = y)$	$x \neq y$

# Logical Operators

- Used for combining **logical values** and create new logical values.
- Logical **AND** (`&&`)
  - return `true` if both operands are `true`, `false` otherwise.
- Logical **OR** (`||`)
  - return `false` if both operands are `false`, `true` otherwise.
- Logical **NOT** (`!`)
  - Invert the Boolean value of the operand.

x	y	x&&y
true	true	true
true	false	false
false	true	false
false	false	false

x	y	x  y
true	true	true
true	false	true
false	true	true
false	false	false

x	!x
true	false
false	true

# Do not mix == and =

```
x=0;  
y=1;  
if  (x = y) {  
    cout << "x and y are equal";  
}  
else  
    cout << "unequal";
```

Output:**x and y are equal.**

The expression **x = y** :

- Assign the value of  $y$  to  $x$  ( $x$  becomes 1).
- The value of this expression is the value of  $y$ , i.e. 1 (which represent true)
  - false is represented by 0 .
  - true is represented by non-zero.

# Assignment (=) and Equality Operator (==)

- Example: `x = 1`
- Assignment operator
- Place the value of the variable on the right to the variable on the left.
- The value of this expression will always equal to the value on the right (`1` in this case).

- Example : `x==1`
- Equality operator
- `true` (evaluates to `1`)
  - the value of `x` is `1`.
- `false` (evaluates to `0`)
  - values of `x` is not `1`.
- No space between the two '`=`'

# Relational, Equality & Logical Operators (Summary)

# Relational operators

- Less than  $<$
  - Greater than  $>$
  - Less than or equal to  $\leq$
  - Greater than or equal to  $\geq$

# Equality operators

- Equal to ==
  - Not equal to !=

# Logical operators

- (Unary) negation (i.e., not) !
  - Logical **and** & &
  - Logical **or** ||

*PS:* Expressions with above operators have a *true* or *false* value.

# Decision and Action

## Real life

We make decision almost everyday.  
Decision will be followed by one or more actions.

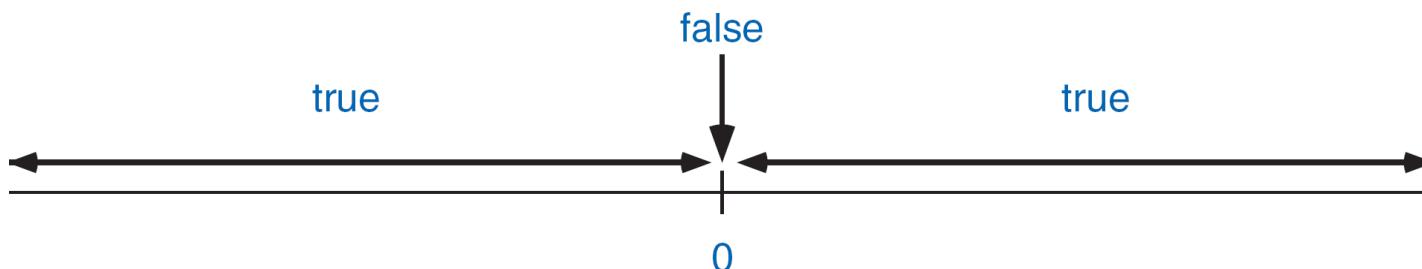


## Programming

Decision is based on logical expression.  
Action is in the form of program statement.

# Logical Expression and Operators

- Logical expression can be **true** or **false** only.
  - $x==3$
  - $y==x$
  - $x>10$
- In C++, any **non-zero** expression will be treated as logical **true**.
  - $3-2$  (true)
  - $1-1$  (false)
  - $x=0$  (false)
  - $X=1$  (true)

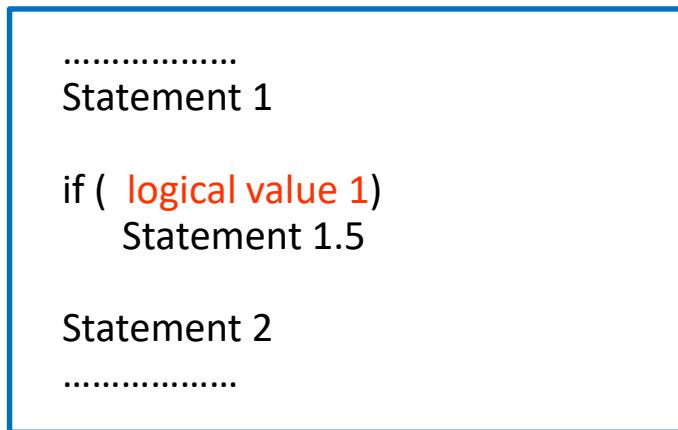


# Conditional Statements

- In decision making process, **logical** value can be used to determine the actions to take.
- E.g.
  - **If** it is raining, bring an umbrella.
  - **If** the canteen is too crowded, go to festival walk for lunch.
- In programming, certain statements will only be executed when certain condition is **fulfilled**. We call them **conditional statements**.

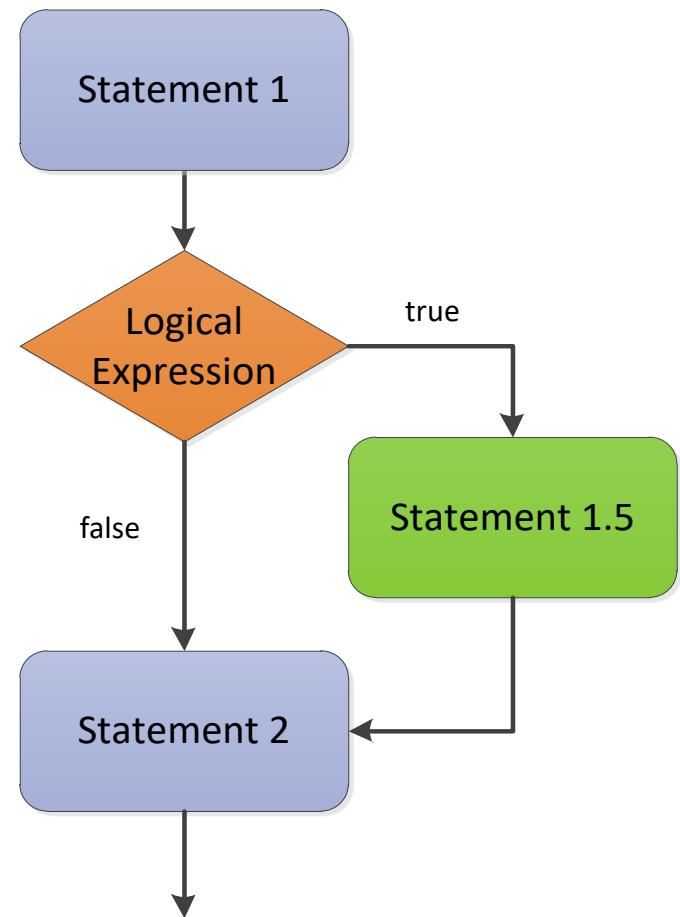
# Conditional Statement: Simplest Form - if

- One or more statements will be executed if the condition is true.



- C++ example:

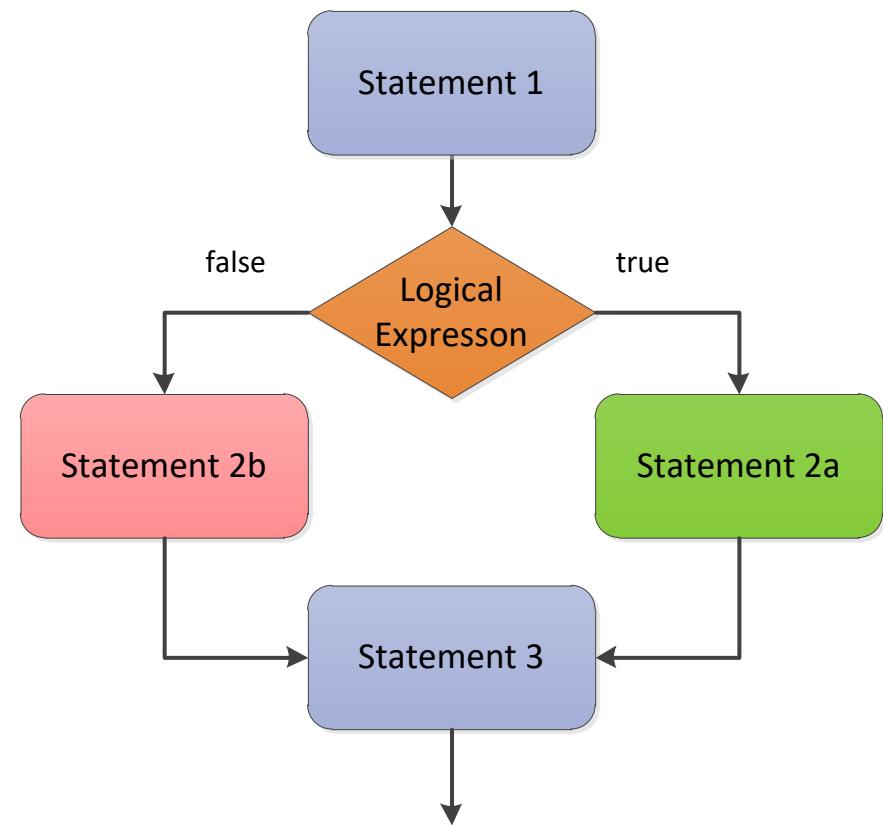
```
cin >>x;  
  
if (x<0)  
    x=-x;  
  
cout << x << endl;
```



# Two-way Selection

- If the condition is **true**, one or more statements will be executed.
- If the condition is **false**, another set of statements will be executed.

```
.....  
Statement 1  
  
if ( Logical expression)  
    statement 2a //action for true  
else  
    statement 2b //action for false  
  
Statement 3  
.....
```



# Some Points to Note

The expression should be enclosed with parenthesis ()

No semi-colon after if or else

The else part is optional

```
if (i == 3)
```

```
a++;
```

```
else
```

```
a--;
```

The semicolons belong to the expression statements, not to the if ... else statement

i == 3 evaluates to a non-zero value

i == 3 evaluates to zero

If i==3 is true, increment a by 1, otherwise (i==3 is false), decrement a by 1.

# Compound Statement

- Group statements into an executable unit (a **block**):
  - if there are more than one statement to be executed for a particular `if`, `else if`, or `else` statement.

```
if (mark>=70) {  
    cout << "You get grade A.\n";  
    cout << "Excellent!\n";  
}  
else if .....  
else .....
```

# Compound Statement

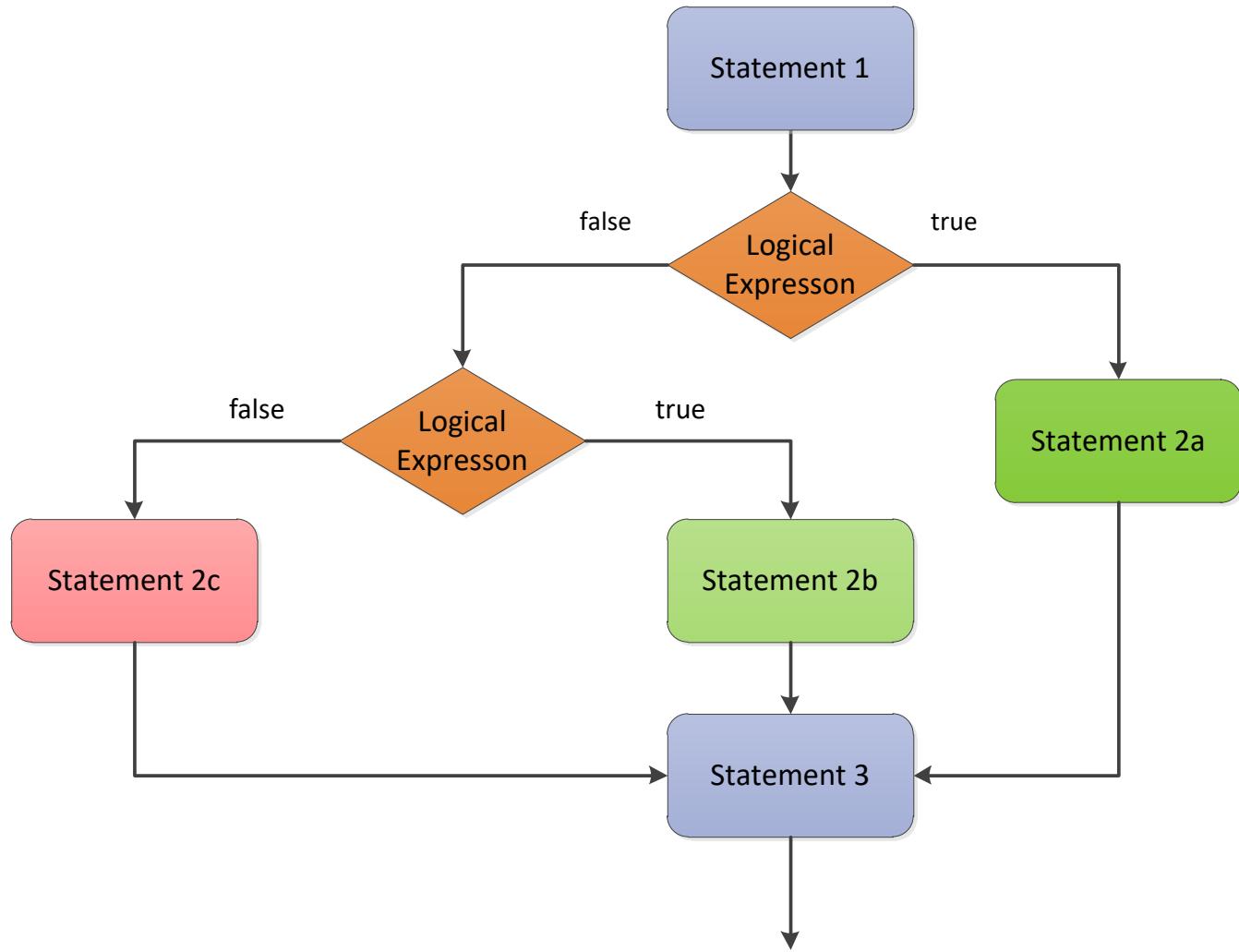
We may group multiple statements to form a compound statement using a pair of braces {}.

```
if (j !=3)
{
    b++;
    cout << b;
} //if
else
    cout < j;
```

Compound statements are treated as one statement.

```
if (j !=5 && d==2)
{
    j++;
    d--;
    cout << j << d;
} //if
else
{
    j--;
    d++;
    cout << j << d;
} //end else
```

# Beyond Two Way Condition...Nested

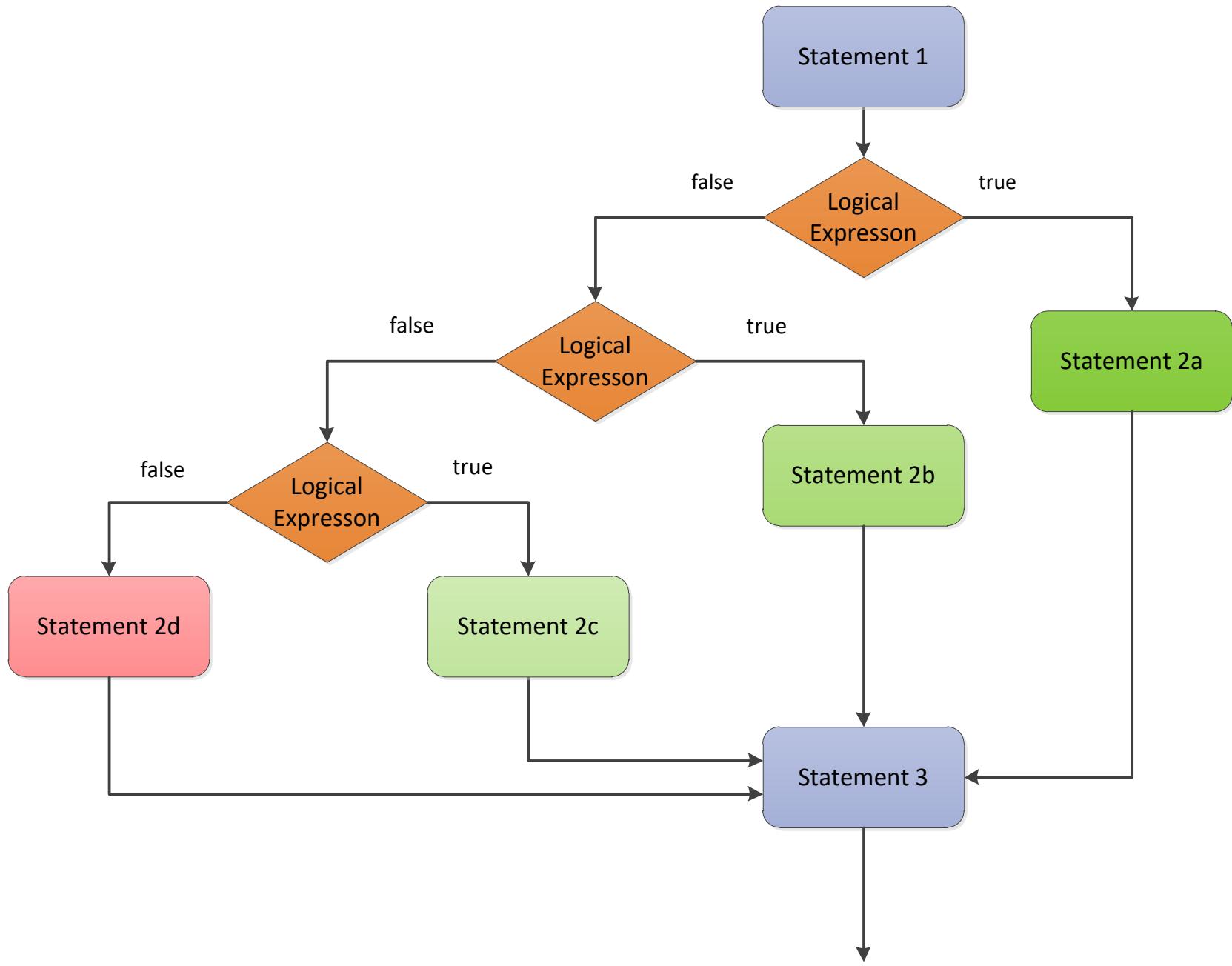


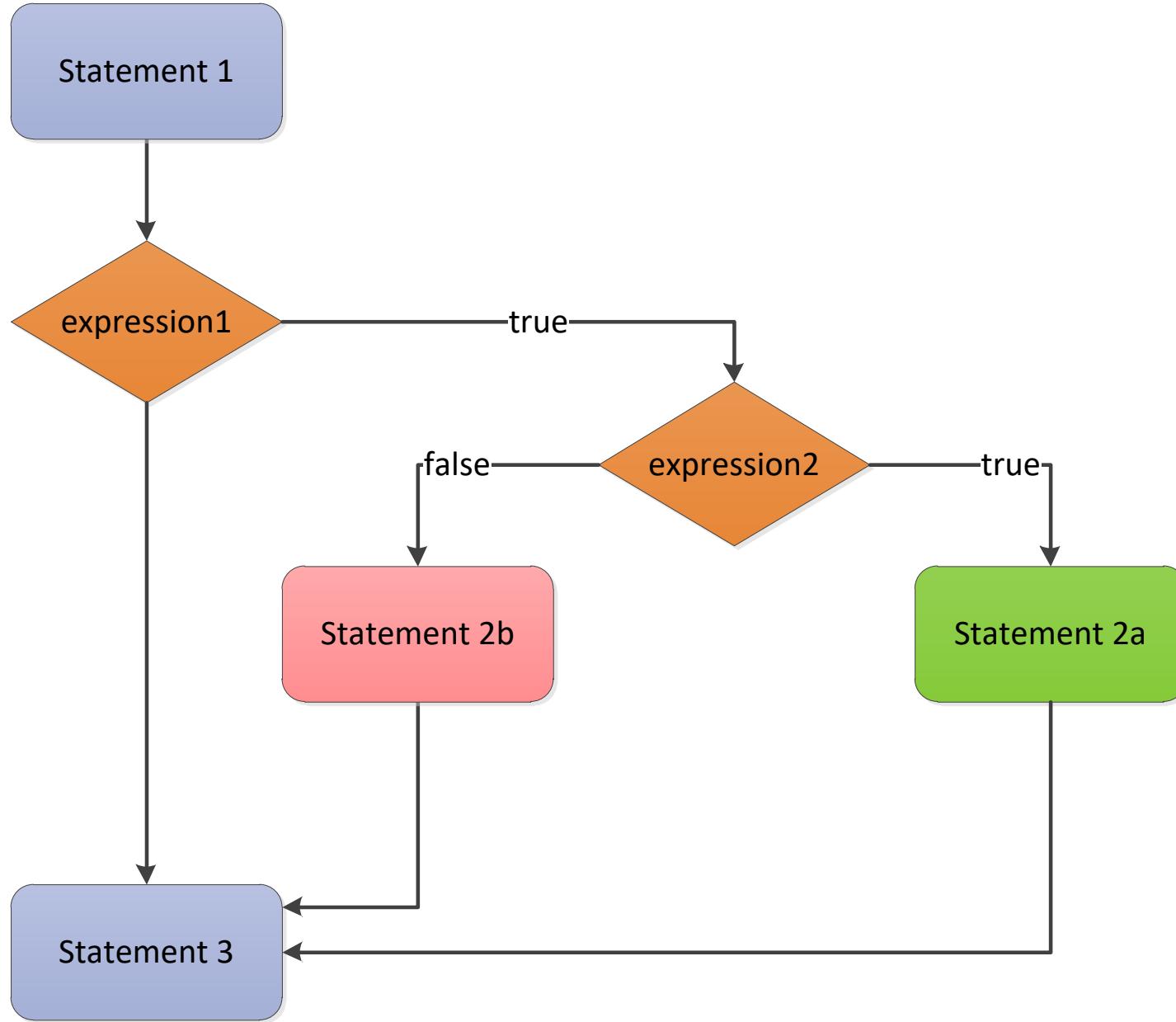
# Beyond Two Way Condition...

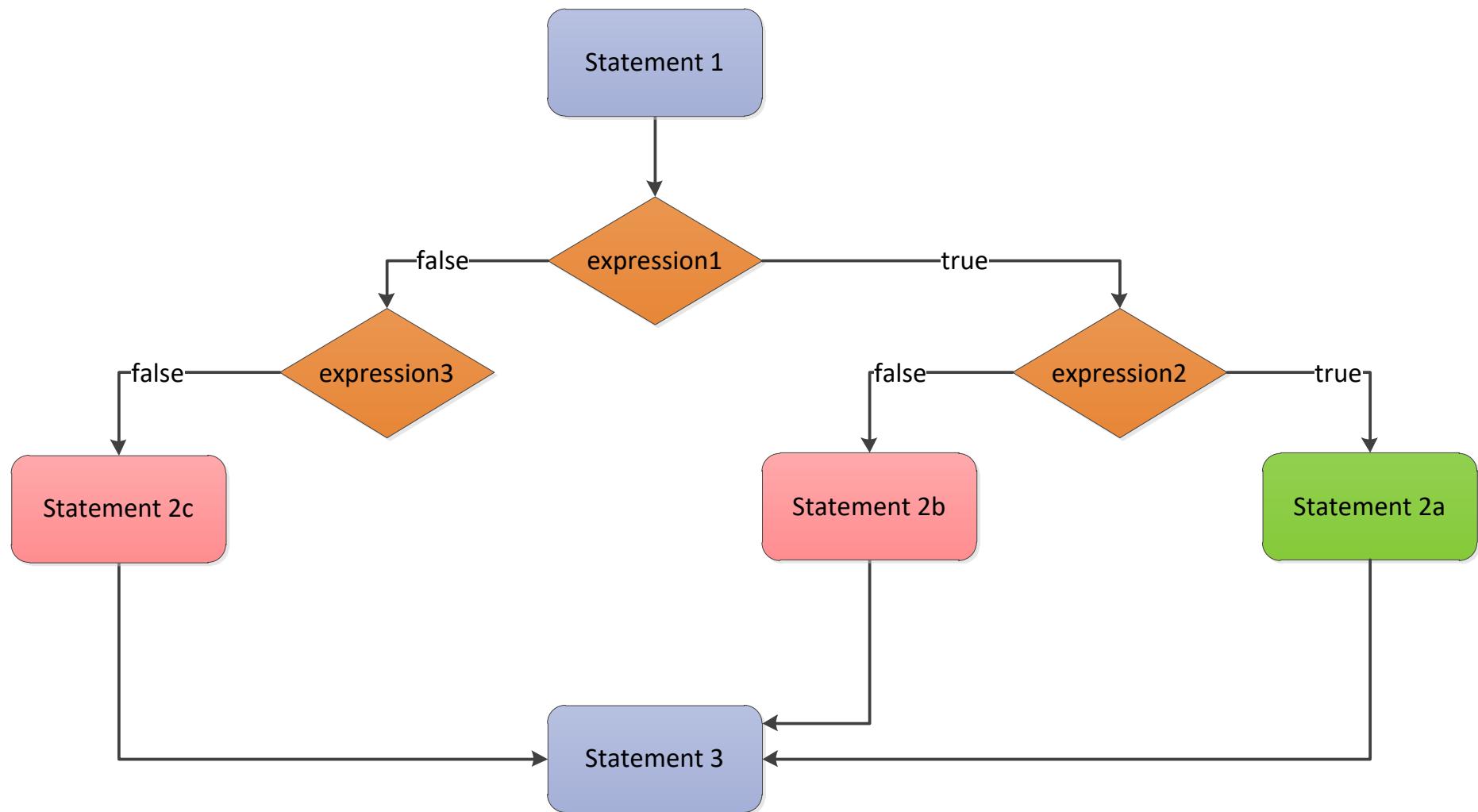
- In C++, **conditional statements** has the following format:

```
if ( logical value 1 ) {  
    statements for logical value 1 is true  
} else if (logical value 2) {  
    statements for logical value 1 is false but logical value 2 is true  
} else if (...)  
...  
} else {  
    statements for all the logical values are false  
}
```

- The `else if` and `else` part are optional.
- The braces can be omitted if the block contains only **one** statement.







# Examples – Single Statement

```
if (x>5) {  
    cout << " x is too large";  
}
```

```
if (x>5)  
    cout << " x is too large";  
else if (x<3)  
    cout << " x is too small";
```

```
if (x>5)  
    cout << " x is too large";  
else if (x<3)  
    cout << " x is too small";  
else  
    cout << " x is a valid answer";
```

# Examples – Compound Statements

```
if (x==3)
    cout << "x is equal to 3";

if (x>y) {
    z=x-y;
    cout << "x is larger, the difference is " << z;
} else {
    z=y-x;
    cout << "y is larger, the difference is " << z;
}
```

# Beware of Empty Statements!

```
int x=5;  
if (x!=5);  
x=3;  
cout << x;  
/*output is 3*/
```

```
int x=5;  
if (x!=5)  
x=3;  
cout << x;  
/*output is 5*/
```

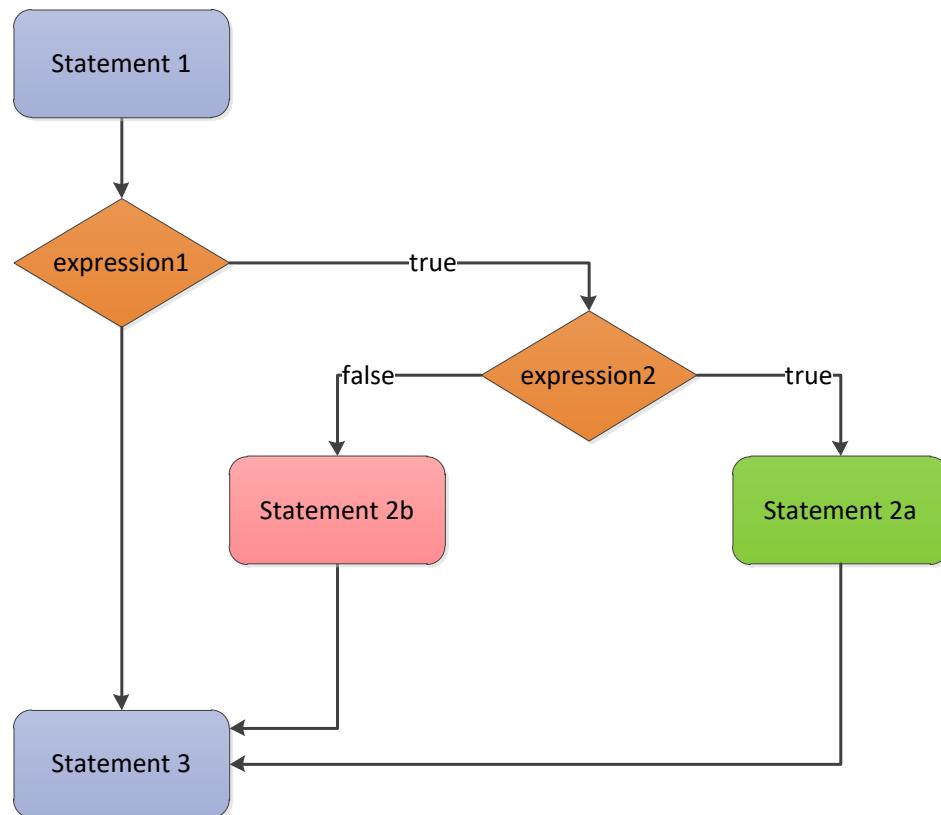
An empty statement can be specified by a semi-colon '`;`'. Empty statement specifies that no action should be performed.

For the second program, `x` is assigned 3 if `x` not equals to 5.

For the first program, because of the extra semi colon at the end of the if statement, nothing is executed when `x != 5`.

# Nested if Statement

An if-else statement is included by another if-else statement.



```
statement1
if (expression1)
    if (expression2)
        statement2a
    else
        statement2b
statement3
```

# Dangling else Problem

- With which if the else part is associated?

```
if (a==1)
    if (b==2)
        cout << "***\n";
else
    cout << "###\n";
```

```
if (a==1)
    if (b==2)
        cout << "***\n";
else
    cout << "###\n";
```

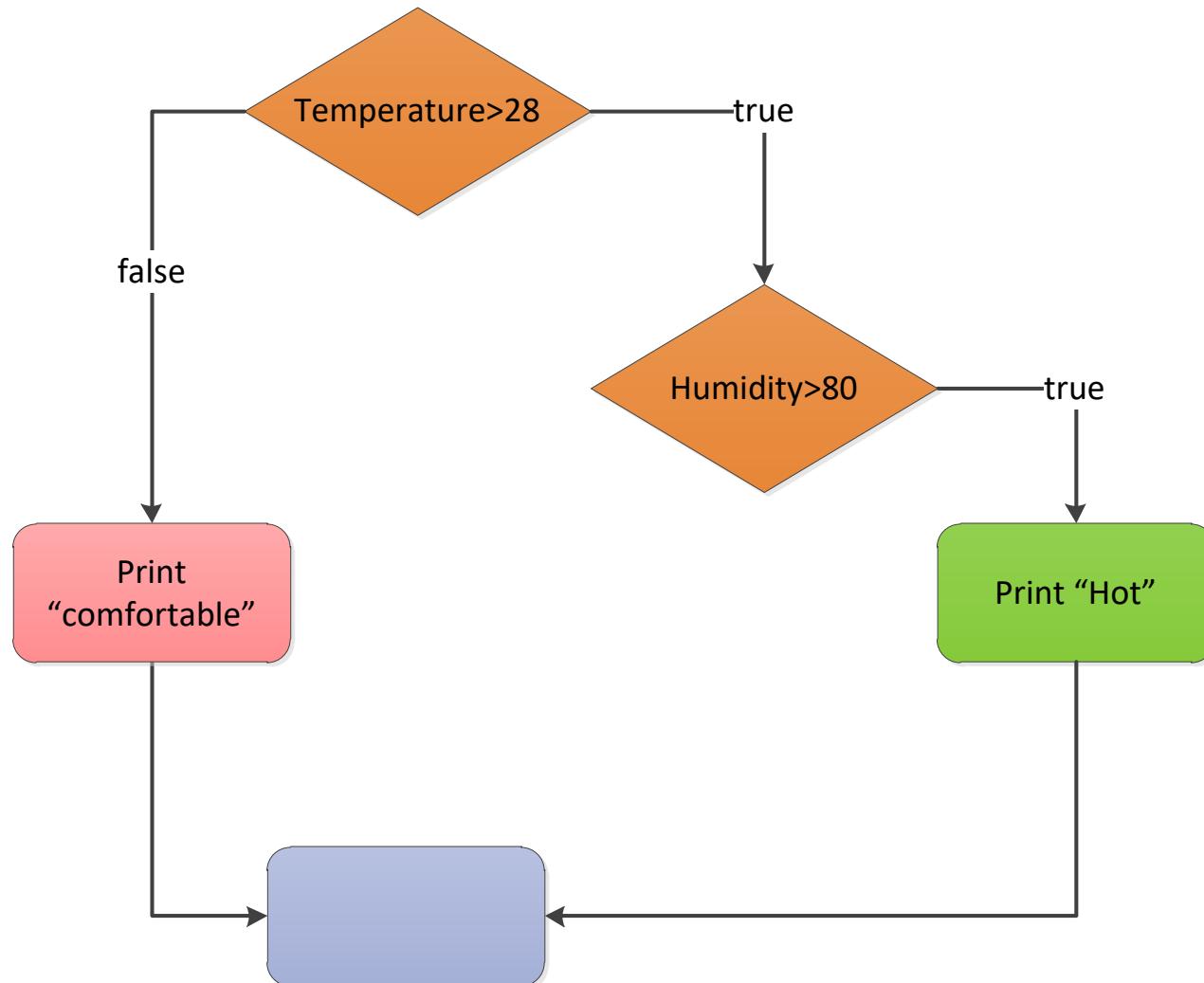
# Dangling else Problem

- An else attached to the nearest if.

```
if (a==1)
    if (b==2)
        cout << "***\n";
else
    cout << "###\n";
```

```
if (a==1)
    if (b==2)
        cout << "***\n";
else
    cout << "###\n";
```

# Suppose We Want to Implement the Following Logic



# Condition

- Certain portion of code is executed when certain condition is true .
- The condition is specified by an **expression**, which evaluates to true or false.

# Example: Passing CS2313

- If you get a total mark greater than or equal to 34. You will pass the course.
  - *This is just an example!*
- Otherwise, you will fail.

Greater than or equal to is a kind of “relational operator”.

Represented by the operator `>=` in C++.

How to represent the above logic in C++?

# Example 1a: Pass or Fail?

```
int mark;  
cout << "What is your mark?\n";  
cin >> mark;  
if (mark>=34)  
    cout << "You passed in CS2313!\n";
```

The condition should be enclosed within () .  
If the input mark is greater than or equal to 34,  
the blue statement is executed.

# Example 1b: Pass or Fail?

```
int mark;  
cout << "What is your mark?\n";  
cin >> mark;  
if (mark>=34){  
    cout << "You passed in CS2313!\n";  
    cout << "congratulations\n";  
}
```

If more than one statements are specified within an `if` statement, group the statements in a pair of braces `{ }`.

# Example 1c: Pass or Fail?

```
int mark;  
cout << "What is your mark?\n";  
cin >> mark;  
if (mark>=34){  
    cout << "You passed in CS2313!\n";  
    cout << "Congratulations\n";  
}  
else  
    cout << "You failed in CS2313!\n";
```

The **else** statement is executed when the condition `mark>=34` is false.

# Example 1c: Pass or Fail?

```
int mark;  
cout << "What is your mark?\n";  
cin >> mark;  
if (mark>=34){  
    cout << "You passed in CS2311!\n";  
    cout << "Congratulations\n";  
}  
else  
    cout << "You failed in CS2311!\n";  
cout << "You should retake the course\n";
```

Suppose the user inputs 35. The output:

You passed in CS2311!

Congratulations

You should retake the course

Why?

# Example 1c: Pass or Fail?

```
int mark;  
cout << "What is your mark?\n";  
cin >> mark;  
if (mark>=34){  
    cout << "You passed in CS2311!\n";  
    cout << "Congratulations\n";  
}  
else {  
    cout << "You failed in CS2311!\n";  
    cout << "You should retake the course\n";  
}
```

Include a brace to group the statements in the **else** part!

# Example 2a: Mark to Grade Conversion

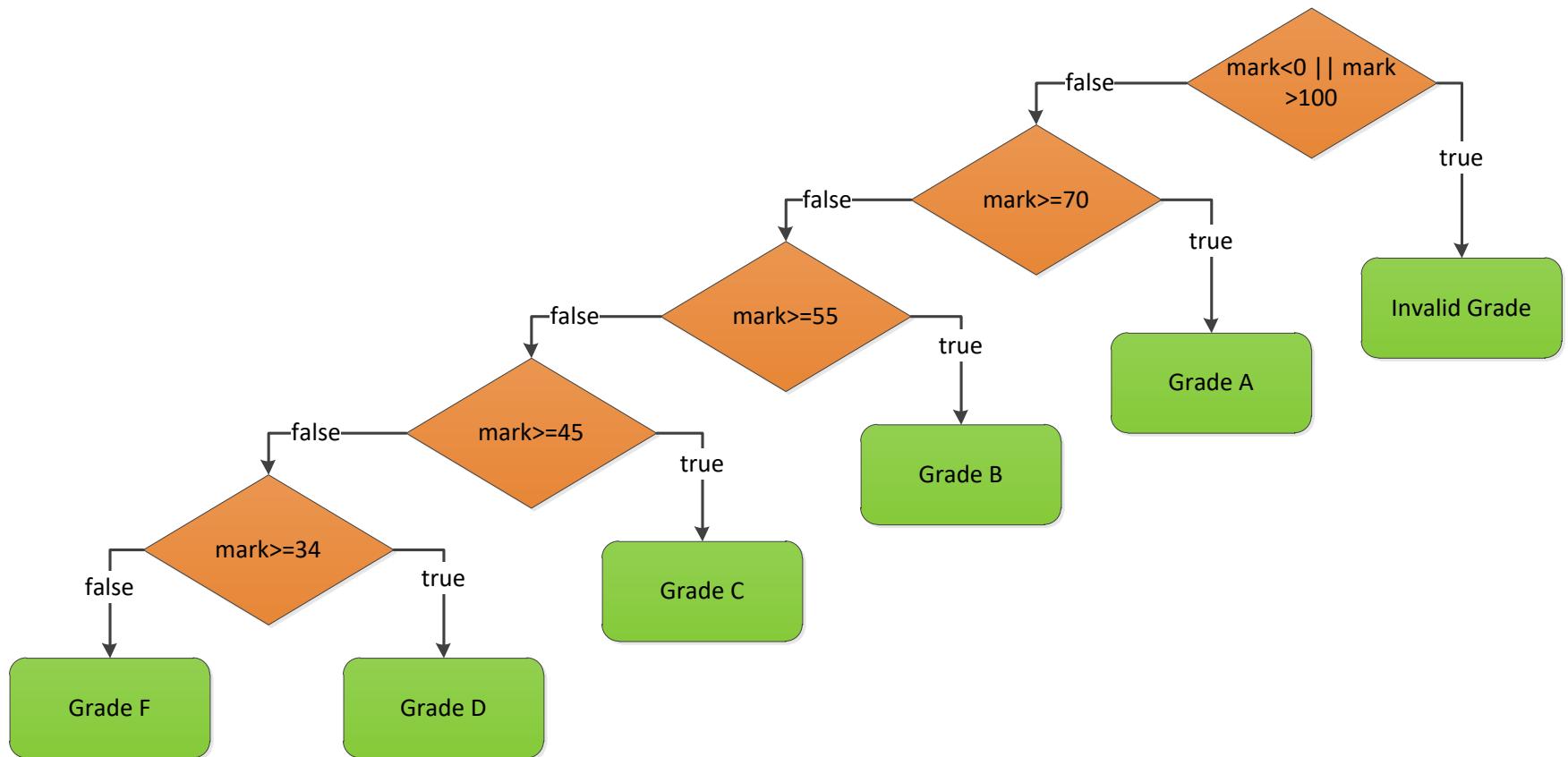
```
if (mark>=70 && mark<=100)
    cout << "A";
if (mark>=55 && mark<70)
    cout << "B";
if (mark>=45 && mark<55)
    cout << "C";
if (mark>=34 && mark<45)
    cout << "D";
if (mark<34 && mark>0)
    cout << "F";
if (mark<0 || mark>100)
    cout << "Invalid Grade";
```

# Mark to Grade Conversion (else-if version)

```
if (mark<0 || mark>100)
    cout << "Invalid Grade";
else if (mark>=70)
    cout << "A";
else if (mark>=55)
    cout << "B";
else if (mark>=45)
    cout << "C";
else if (mark>=34)
    cout << "D";
else
    cout << "F";
```

The else if or else part is executed  
only if all the preceding conditions are false.

# Mark to Grade Conversion



# C++ Syntax Is Different from the Math Syntax

```
if (mark>=70 && mark<=100)  
.....
```

Can we express the above condition as follows?

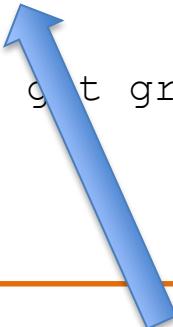
```
if (70<=mark<=100)  
.....
```

Ans: No, the above statement is syntactically correct (can be compiled), but the meaning is **different**.

# Nested if

An if statement can be nested within another if statement.

```
if (mark>=70 && mark <=100) {  
    if (mark>90)  
        cout << "You get grade A+.\\n";  
    else if (mark>80)  
        cout << "You get grade A.\\n";  
    else  
        cout << You g t grade A-.\\n";  
}  
else if .....
```



This statement is executed if  $\text{mark} \geq 70$  and  $\text{mark} \leq 100$  and  $\text{mark} \leq 90$  and  $\text{mark} > 80$ .

## Example 2b

Modify the previous program such that if the mark is 100, the statement "Full mark!" should be printed (in addition to the grade).

# Example 2b

```
if (mark>=70 && mark <=100) {  
    if (mark>90) {  
        cout << "You get grade A+.\\n";  
        if (mark==100)  
            cout << "\\nFull mark!\\n";  
    }  
    else if (mark>80)  
        cout << "You get grade A.\\n";  
    else  
        cout << "You get grade A-.\\n";  
}  
else if .....
```

# Short-Circuit Evaluation

Evaluation of expressions containing `&&` and `||` **stops** as soon as the outcome `true` or `false` is known and this is called *short-circuit evaluation*.

Short-circuit evaluation can improve program efficiency.

Short-circuit evaluation exists in some other programming languages too, e.g., C and Java.

# Short-Circuit Evaluation

Given integer variable `i`, `j` and `k`, what are the **outputs** when running the program fragment below?

```
k = (i=2) && (j=2);
cout << i << j << endl; /* 2 2 */
k = (i=0) && (j=3);
cout << i << j << endl; /* 0 2 */
k = i || (j=4);
cout << i << j << endl; /* 0 4 */
k = (i=2) || (j=5);
cout << i << j << endl; /* 2 4 */
```

# switch Statement

General format of switch statement (*selection statement*).

```
switch (expression) {  
    case constant-expr1: statement1  
    case constant-expr2: statement2  
    ...  
    ...  
    case constant-exprN: statementN  
    default: statement  
}
```

# Example

```
#include <iostream>
using namespace std;

int main() {
    int x;
    cin >> x;

    switch (x) {
        case 0:
            cout << "Zero";
            break;
        case 1:
            cout << "One";
            break;
        case 2:
            cout << "Two";
            break;
        default:
            cout << "Greater than two";
    } //end switch
    return 0;
}
```

# switch Statement (cont'd)

## Semantics

- Evaluating the switch expression returns an integer type (int, long, short, char).
- Go to the case label having the constant value that matches the value of the switch expression; if a match is not found, go to the default label; if default label does not exist, terminate the switch.
- Terminate the switch when a break statement is encountered.
- If there is no break statement, execution “*falls through*” to the next statement in the succeeding case.

# A Program Segment Using switch

```
while ((c = getchar()) != EOF) { /* get a char */
    switch (c) {
        case '0': case '1': case '2': case '3': case '4':
        case '5': case '6': case '7': case '8': case '9':
            digit_count++;
            /* no braces is needed */
            break;
        case ' ': case '\n': case '\t':
            white_character_count++;
            break;
        default:
            other_character_count++;
            break;
    }
}
```

```

#include <iostream>
using namespace std;

int main() {
    cout << "Difficulty Levels" << endl;
    cout << "1-easy"    << endl;
    cout << "2-normal" << endl;
    cout << "3-hard"   << endl;

    int choice;
    cout << "Choice: ";
    cin  >> choice;

    switch (choice)
    {
        case 1:
            cout << "you picked easy" << endl;
            break;
        case 2:
            cout << "you picked normal" << endl;
            break;
        case 3:
            cout << "you picked hard" << endl;
            break;
        default:
            cout << "You made an illegal choice" << endl;
    }

    return 0;
}

```

C:\Windows\system32\cmd.exe

Difficulty Levels  
 1-easy  
 2-normal  
 3-hard  
 Choice: 1  
 you picked easy  
 Press any key to continue . . . -

C:\Windows\system32\cmd.exe

Difficulty Levels  
 1-easy  
 2-normal  
 3-hard  
 Choice: 2  
 you picked normal  
 Press any key to continue . . . -

C:\Windows\system32\cmd.exe

Difficulty Levels  
 1-easy  
 2-normal  
 3-hard  
 Choice: 3  
 you picked hard  
 Press any key to continue . . .

Select C:\Windows\system32\cmd.exe

Difficulty Levels  
 1-easy  
 2-normal  
 3-hard  
 Choice: 6  
 You made an illegal choice  
 Press any key to continue . . . -

```
#include <iostream>
using namespace std;

int main() {
    cout << "Difficulty Levels" << endl;
    cout << "1-easy"    << endl;
    cout << "2-normal"  << endl;
    cout << "3-hard"    << endl;

    int choice;
    cout << "Choice: ";
    cin  >> choice;

} switch (choice)
{
    case 1:
        cout << "you picked easy" << endl;
    case 2:
        cout << "you picked normal" << endl;
    case 3:
        cout << "you picked hard" << endl;
    default:
        cout << "You made an illegal choice" << endl;
}

return 0;
}
```

# Without **break**

```
C:\Windows\system32\cmd.exe
```

```
Difficulty Levels
1-easy
2-normal
3-hard
Choice: 1
you picked easy
you picked normal
you picked hard
You made an illegal choice
Press any key to continue . . . ■
```

```
C:\Windows\system32\cmd.exe
```

```
Difficulty Levels
1-easy
2-normal
3-hard
Choice: 3
you picked hard
You made an illegal choice
Press any key to continue . . . ■
```

```
C:\Windows\system32\cmd.exe
```

```
Difficulty Levels
1-easy
2-normal
3-hard
Choice: 2
you picked normal
you picked hard
You made an illegal choice
Press any key to continue . . . ■
```



# Conditional (?:) Operator

- General format of the ternary ?: operator is

```
expr1 ? expr2 : expr3
```

- Semantics**
  - **expr1** is evaluated.
  - If the above result is non-zero, then **expr2** is evaluated; else **expr3** is evaluated.
  - The value of the whole ?: expression is the value of expression evaluated at the end.
  - Data type of the returning value is determined by both **expr2** and **expr3**, but not the one being evaluated ultimately.
- Example**
  - `int min_x=(x>y)?y:x;`

# Common Errors in Conditional Statements

## Common Error 1: Forgetting Necessary Braces

```
if (radius >= 0)
    area = radius * radius * PI;
    cout << "The area "
        << " is " << area;
```

(a) Wrong

```
if (radius >= 0)
{
    area = radius * radius * PI;
    cout << "The area "
        << " is " << area;
}
```

(b) Correct

# Common Errors in Conditional Statements

## Common Error 2: Wrong Semicolon at the if Line

Logic Error

```
if (radius >= 0);  
{  
    area = radius * radius * PI;  
    cout << "The area "  
        << " is " << area;  
}
```

(a)

Equivalent

---

---

Empty Body

```
if (radius >= 0) {};  
{  
    area = radius * radius * PI;  
    cout << "The area "  
        << " is " << area;  
}
```

(b)

# Common Errors in Conditional Statements

## Common Error 3: Mistakenly Using = for ==

```
if (count = 1)
    cout << "count is zero" << endl;
else
    cout << "count is not zero" << endl;
```

# Common Errors in Conditional Statements

## Common Error 4: Redundant Testing of Boolean Values

```
if (even == true)  
    cout << "It is even.";
```

(a)

Equivalent

```
if (even)  
    cout << "It is even.";
```

(b)

This is better

**This is not an error!**

# Trace if-else statement

Suppose score is 70.0

The condition is false

```
if (score >= 90.0)
    cout << "Grade is A";
else if (score >= 80.0)
    cout << "Grade is B";
else if (score >= 70.0)
    cout << "Grade is C";
else if (score >= 60.0)
    cout << "Grade is D";
else
    cout << "Grade is F";
```

This is just an example!

# Trace if-else statement

Suppose score is 70.0

The condition is false

```
if (score >= 90.0)
    cout << "Grade is A",
else if (score >= 80.0)
    cout << "Grade is B";
else if (score >= 70.0)
    cout << "Grade is C";
else if (score >= 60.0)
    cout << "Grade is D";
else
    cout << "Grade is F";
```

# Trace if-else statement

Suppose score is 70.0

The condition is true

```
if (score >= 90.0)
    cout << "Grade is A";
else if (score >= 80.0)
    cout << "Grade is B",
else if (score >= 70.0)
    cout << "Grade is C";
else if (score >= 60.0)
    cout << "Grade is D";
else
    cout << "Grade is F";
```

# Trace if-else statement

Suppose score is 70.0

grade is C

```
if (score >= 90.0)
    cout << "Grade is A";
else if (score >= 80.0)
    cout << "Grade is B";
else if (score >= 70.0)
    cout << "Grade is C";
else if (score >= 60.0)
    cout << "Grade is D";
else
    cout << "Grade is F";
```

# Trace if-else statement

Suppose score is 70.0

Exit the if statement

```
if (score >= 90.0)
    cout << "Grade is A";
else if (score >= 80.0)
    cout << "Grade is B";
else if (score >= 70.0)
    cout << "Grade is C";
else if (score >= 60.0)
    cout << "Grade is D";
else
    cout << "Grade is F";
```

# Summary

- **Boolean** logic has two values only: true or false.
- **Conditional** statements are the statements will only be executed under certain condition.
- In C++, there are two approaches to construct conditional statement:
  - `If (...){...}else{...} .`
  - `switch (...) {...case:break} .`