CS2311 Computer Programming

LT4: Flow Control

Conditional Statements

Sequential Flow of Control

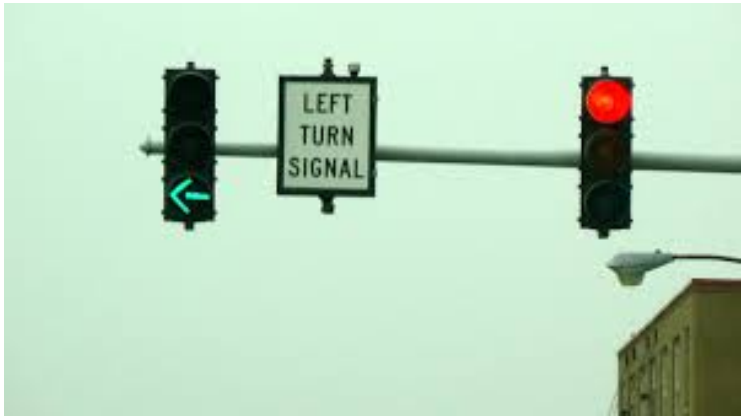
- So far, the control flow is completely sequential
  - Program always executes the same sequence of statements from the start of the program to the end

But

- In some situations, we want to our program to produce different effects based on different program conditions
  - In other words, we want our code to produce different sequences of statements

Decision and Action

- We make decisions everyday
  - AC-1 canteen? AC-2? AC-3?
- Decision will be followed by one or more action(s)
- In programming:
  - Decision is based on conditions, e.g., logical expressions
  - Action is in the form of programming statements



Comparison/Relational 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

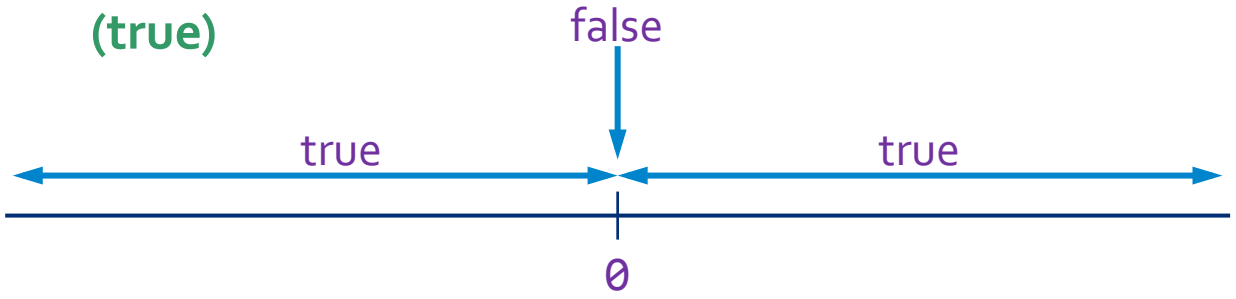
Logical Expressions and Operators

- Logical expressions can be true or false only

```
x == 3
y == x
x > 10
```

- In C++, any expression that evaluates to a non-zero value will be treated as logical true

```
3 - 2       (true)
1 - 1       (false)
x = 0       (false)
X = 1       (true)
```



Assignment = and Equality == Operators

- 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)

- Example:

  `x == 1`
- Equality operator
- T (evaluates to 1)
  - values of the value of x is 1
- F (evaluates to 0)
  - values of x is not 1
- No space between the two '='

Note.
Avoid to compare == of two floating point numbers

Logical Operators

- Used to assess logical expressions
- Logical AND **&&**
  - Both conditions have to be satisfied to be true
- Logical OR **||**
  - Any one condition has to be satisfied to be true
- Logical NOT **!**
  - Invert the logical result of the expression/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

The Logical NOT Operator

- Logical-NOT (`!`) is a unary operator (it only takes one operand). So how does it work?

- To check if age is not between 18 and 65, we simply write:

```
!(age >= 18 && age <= 65)
```

- The Logical-NOT operator only takes one operand, evaluates the operand to `true` or `false`, then flips the result

Effect of Logical NOT Operator

Original Expression	Equivalent Expression
`!(x < y)`	`x >= y`
`!(x > y)`	`x <= y`
`!(x != y)`	`x == y`
`!(x <= y)`	`x > y`
`!(x >= y)`	`x < y`
`!(x == y)`	`x != y`

`!(A && B)` is the same as `(!A) || (!B)`

`!(A || B)` is the same as `(!A) && (!B)`

Summary: Relational, equality & logical operators

- Relational operators
  - less than `<`
  - greater than `>`
  - less than or equal to `<=`
  - greater than or equal to `>=`
- Equality operators
  - equal to `==`
  - not equal to `!=`
- Logical operators
  - logical not `!`
  - logical and `&&`
  - logical or `||`

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

Precedence & Associativity of Operators

Operator precedence (high to low)	Associativity
( ) ++(postfix) – (postfix)	Left to right
++(prefix) -- (prefix)	Right to left
* / %	Left to right
+ -	Left to right
< <= > >=	Left to right
== !=	Left to right
&&	Left to right
||	Left to right
?:	Right to left
= += -= *= /=	Right to left
, (comma operator)	Left to right

Conditional Statements

- In a decision making process, logical value can be used to determine what actions to take.

- Examples:
  - If AC2 canteen is too crowded, then go to AC1/AC3 for lunch

- In programming, certain statements will only be executed when certain condition is fulfilled. We call them ***conditional statements***.

Conditional Statement : `if`

- One statement will be executed if the condition is true
- Syntax:

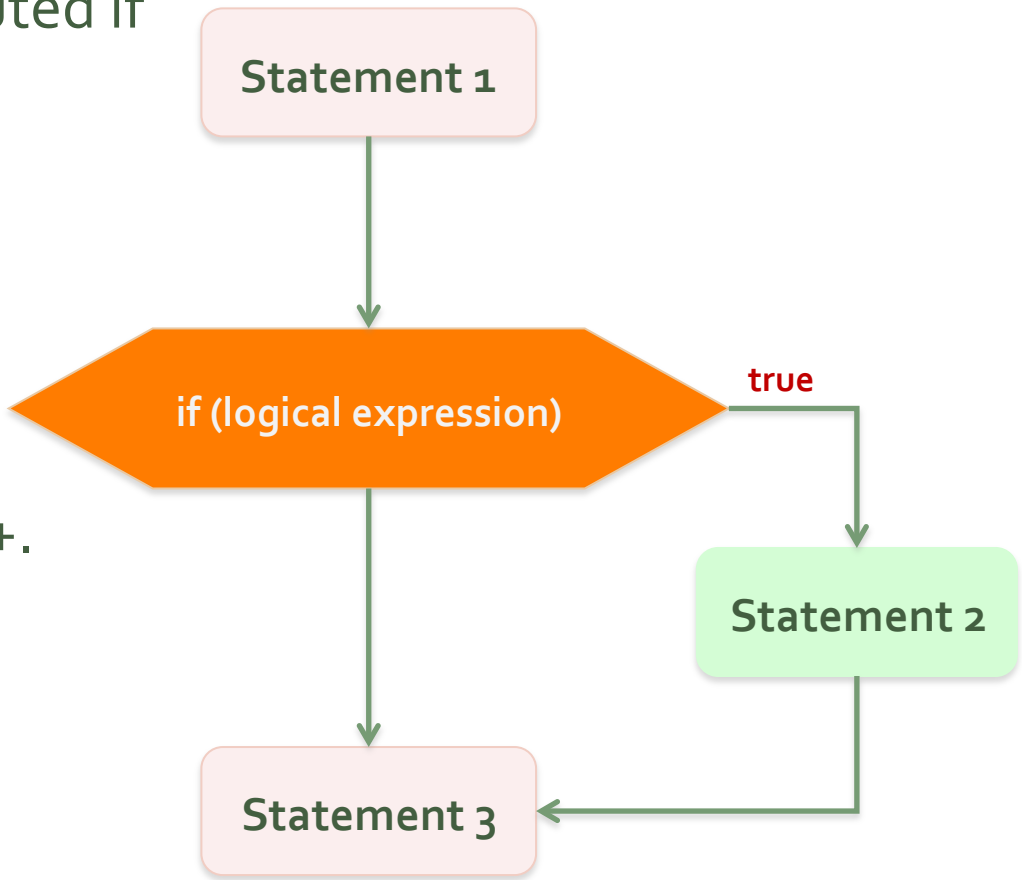
```
if (logical expression)
    statement;
```

- Only one statement will be executed
- This is **one statement** in C++.

```
if (logical expression)
    statement;
```

- C++ example:

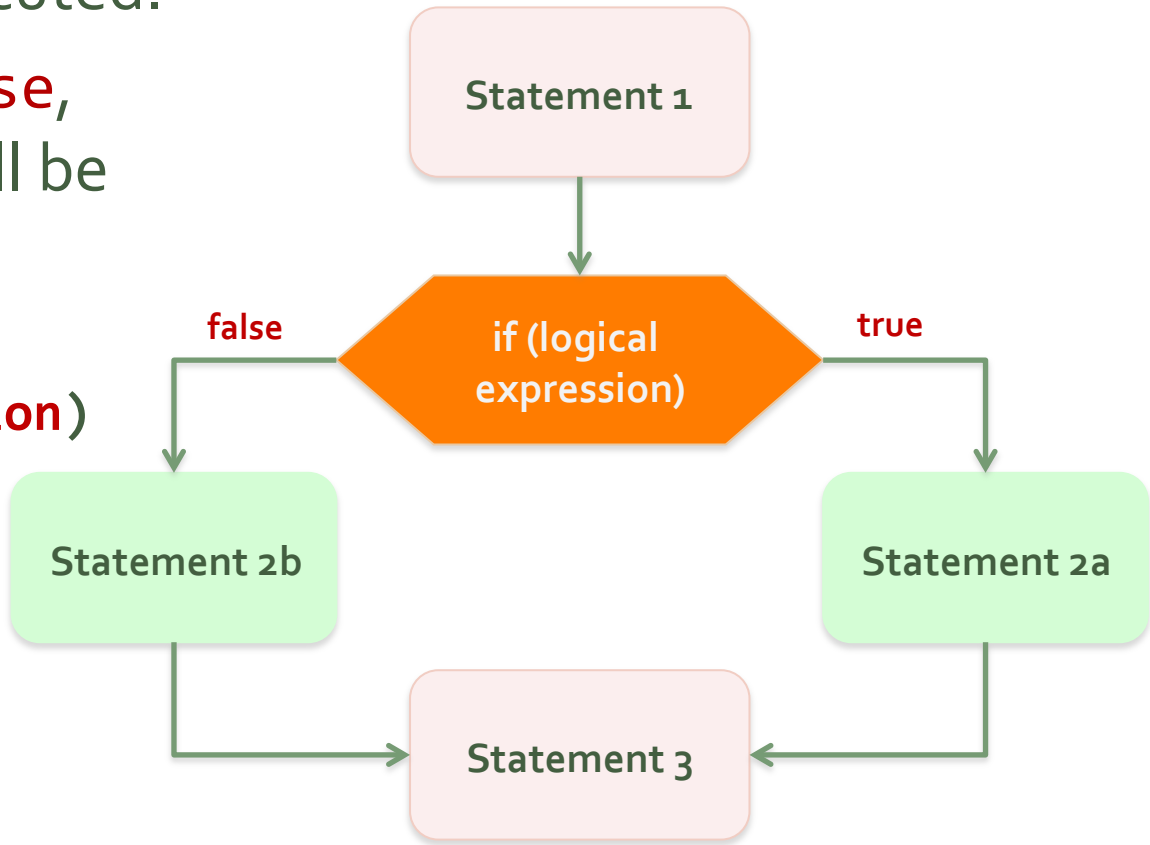
```
cin >> x;
if (x < 0)
  x = -x;
cout  << x << endl;
```



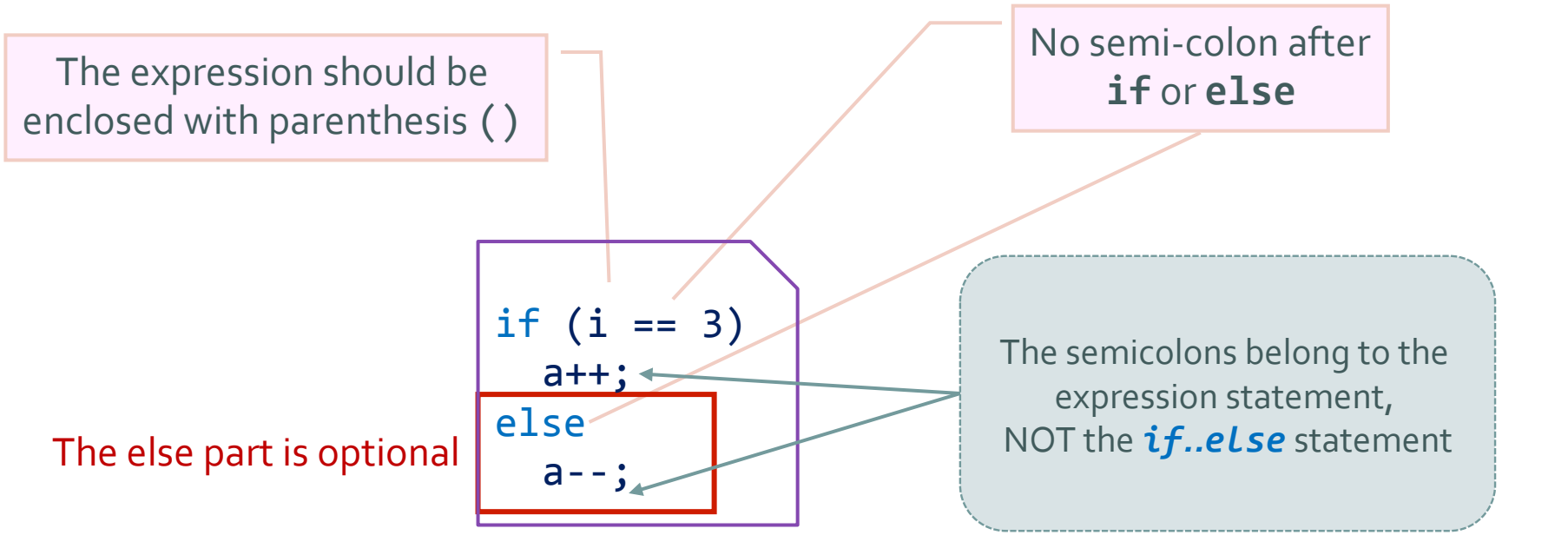
Two-way Selection

- If the condition is `true`, one statement will be executed.
- If the condition is `false`, another statement will be executed

```
.....
if (logical expression)
    //action for true
    statement a;
else
    //action for false
    statement b;
```



Some Points to Note



`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 multiple statements into one block using **{ }** to be executed for a certain **if**, **else if**, or **else** statement
- Must use { } if there're more than one statements for an if, else, or else if statement

```
if (conditional expression) {
    statements_true ... ;
}
```

- This is **one statement** in C++.

```
if (condition) {
    statement;
    statement;
}
```

- C++ Example

```
if (mark >= 90) {
    cout << "You get grade A." << endl;
    cout << "Excellent!" << endl;
}
```

Example 1a: Pass or Fail?

```
int mark;
cout << "What is your mark?" << endl;
cin >> mark;
if (mark >= 34)
   cout << "You passed in CS2311!" << endl;
```

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?" << endl;
cin >> mark;
if (mark >= 34) {
  cout << "You passed in CS2311!" << endl;
  cout << "Congratulations!" << endl;
}
```

If more than 1 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?" << endl;
cin >> mark;
if (mark >= 34) {
  cout << "You passed in CS2311!" << endl;
  cout << "Congratulations!" << endl;
}
else
  cout << "You failed in CS2311…" << endl;
```

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

Example 1c: Pass or Fail?

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

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?" << endl;
cin >> mark;
if (mark >= 34) {
  cout << "You passed in CS2311!" << endl;
  cout << "Congratulations!" << endl;
}
else {
  cout << "You failed in CS2311…" << endl;
  cout << "You should retake the course." << endl;
}
```

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

Compound Statement

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

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

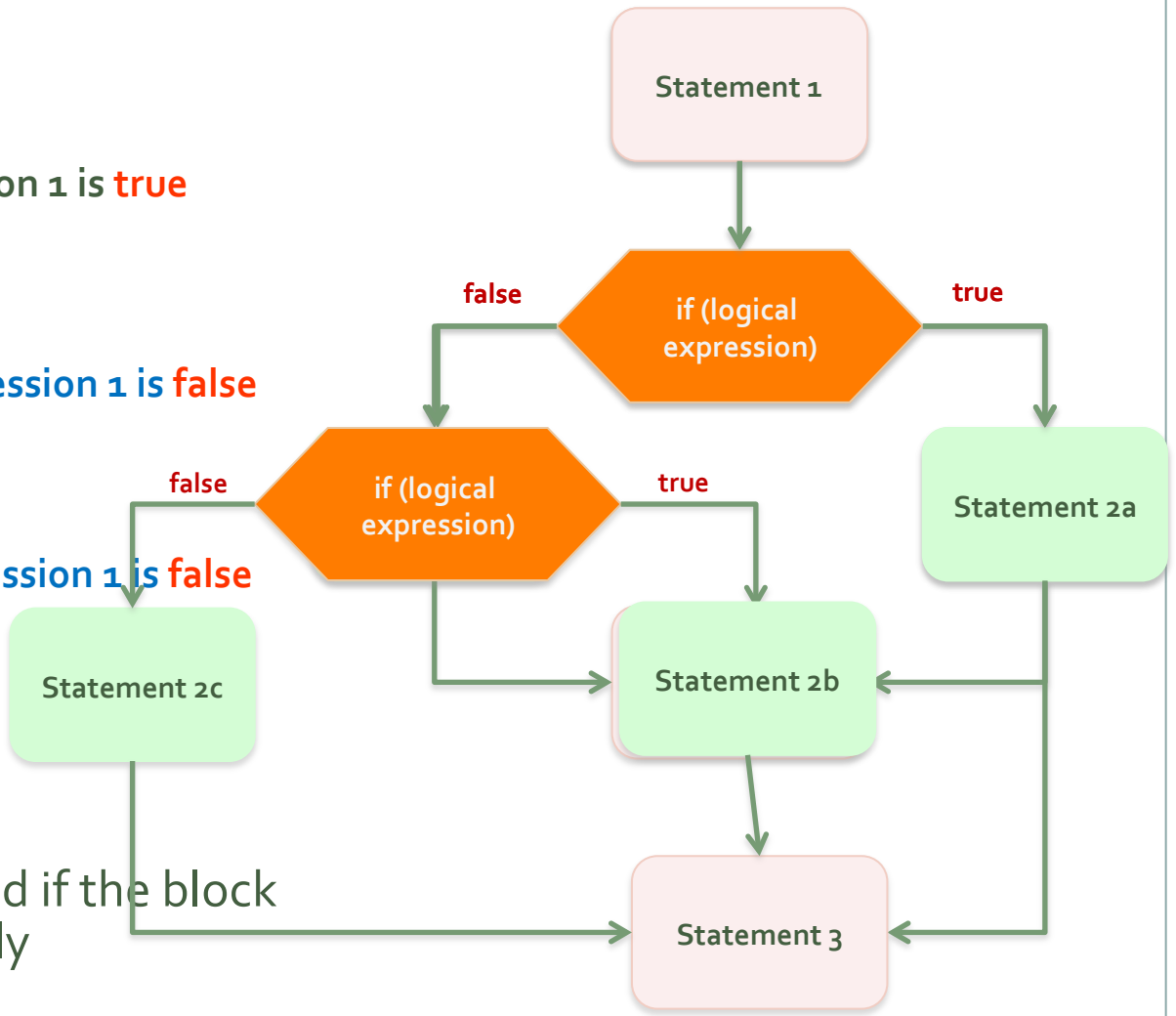
Compound statements are treated as one statement

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

Beyond Two Way Condition…

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

```
statement_1
if (logical expression 1) {
    statements_2a when expression 1 is true
}
else {
    if (logical expression 2) {
        statements_2b when expression 1 is false
    and expression 2 is true
    }
    else {
        statements_2c when expression 1 is false
    and expression 2 is false
    }
}
statement_3
```



- The brackets can be omitted if the block contains one statement only

Multiple `else-if` Statements

- You can have many **nested** "**else if**" statements.

```
.....
if (condition 1)
    //action for true
    statement a;
else if (condition 2)
    //action for true
    statement b;
    .....
else
    //action for false
    statement;
```

An example:

input a score, display grade according to:

A: 100 – 90,

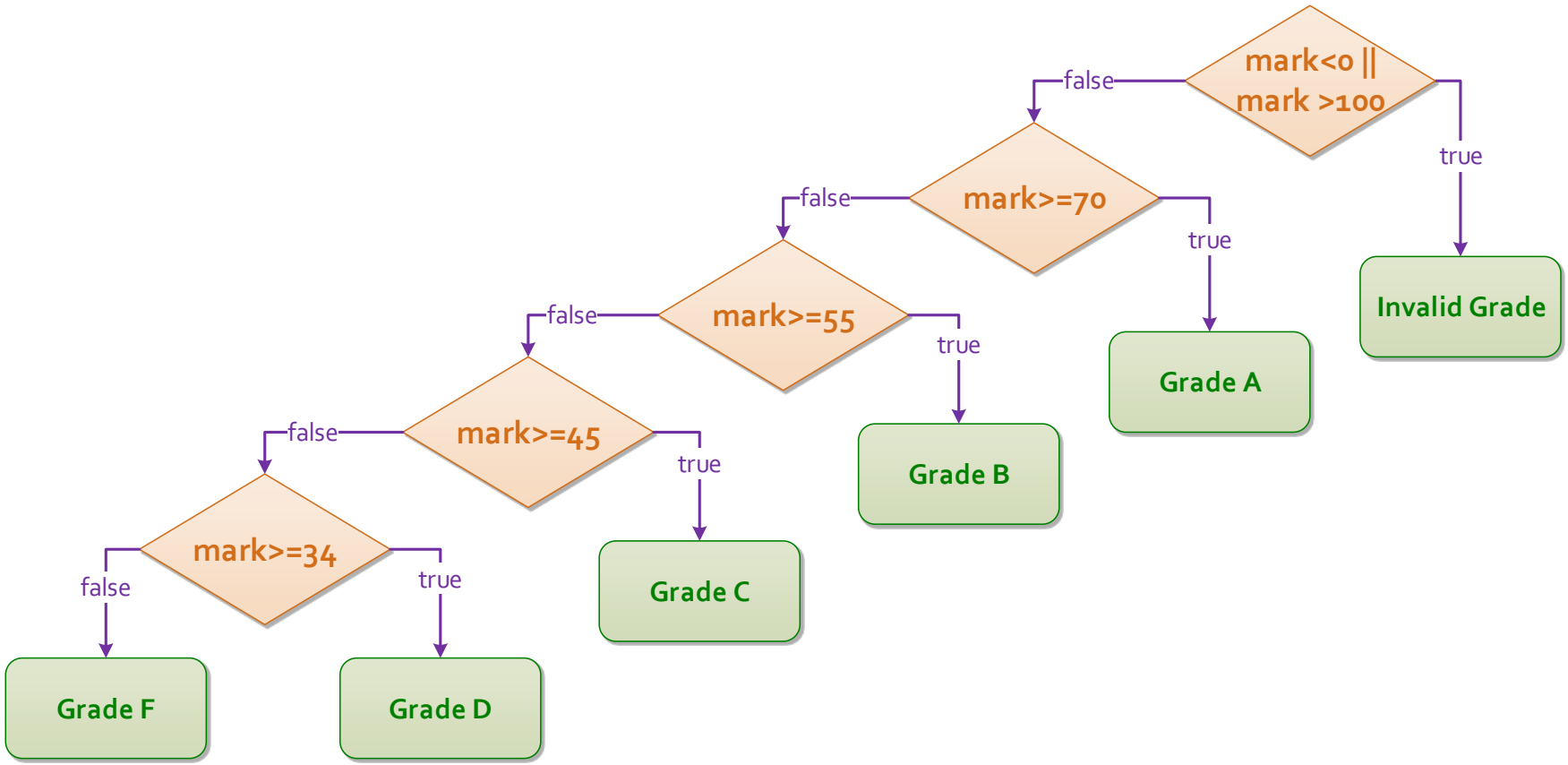
B: 89 – 75,

C: 74 – 55,

D: 54 – 0

```
if (score >= 90)
   grade = 'A';
else if (score >= 75)
   grade = 'B';
else if (score >= 55)
   grade = 'C';
else
   grade = 'D';
```

Mark to Grade Conversion



Mark to Grade Conversion

```
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";
```

```
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";
```

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

Beyond Two Way Condition…

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

```
if (logical expression 1) {
    statements when expression 1 is true
}
else if (logical expression 2) {
    statements when expression 1 is false and expression 2 is true
}
else if (…) {
    …
}
else {
    statements when all the logical expressions are false
}
```

- The ***else if*** and ***else*** part is optional
- The curly brackets/braces **{}** can be omitted if the block contains one statement only

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*/
```

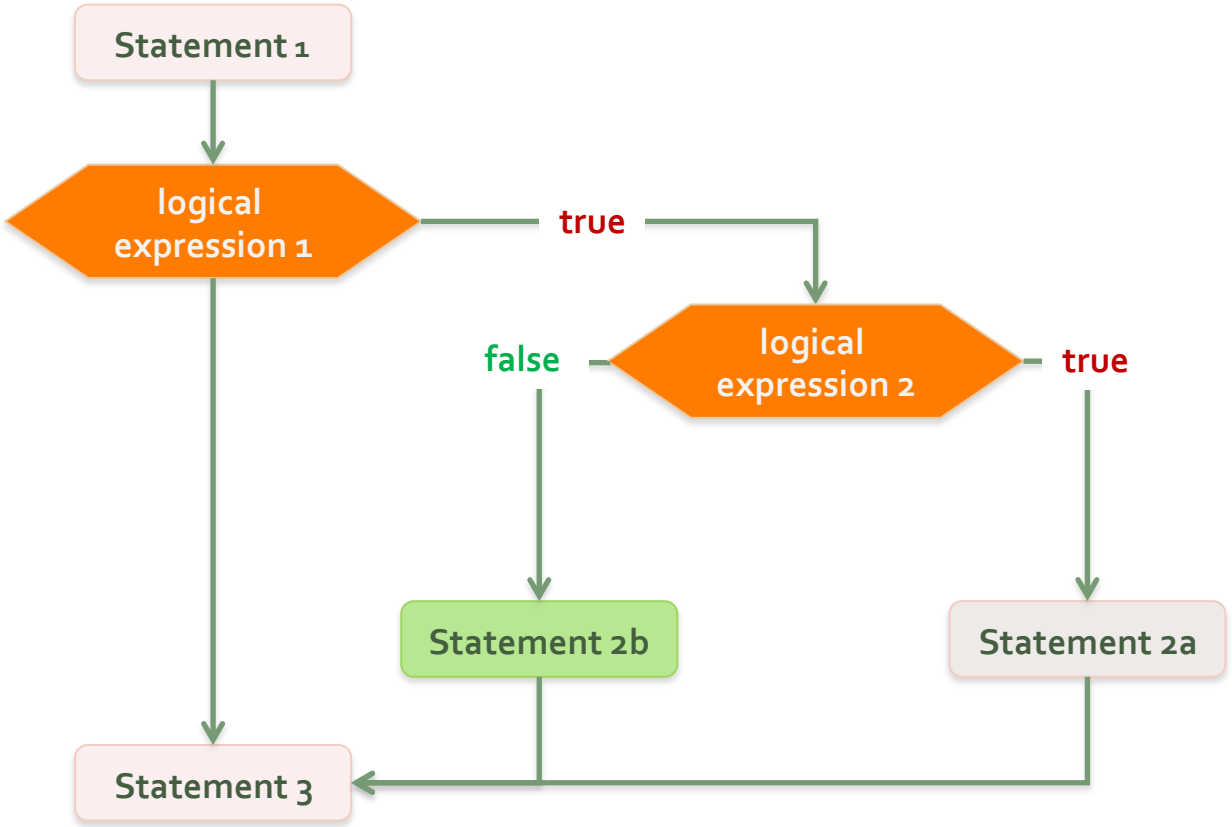
- An empty statement can be specified by a semi-colon **';'**.
- Empty statement specifies that **no action should be performed**.

- For the program on the **right**, `x = 3;` will NOT be executed when x is equal to 5.

- For the program on the **left**, `x = 3;` **will be always executed**.

Nested `if` Statement

An **if-else** statement is included with another **if-else** statement




```
Statement 1
If (expression 1)
    If (expression 2)
        Statement 2a
    else
        Statement 2b
Statement 3
```

Nested `if` Statement

- With which "**if**" the "**else**" part is associated?

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

✗

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

- An "**else**" attached to the nearest "**if**".

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

```
If (a == 1)
   If (b == 2)
      cout << "***";
   else
      cout << "###";
```

Nested if

- An if statement can be nested within another if statement

```
:
if (mark >= 70 && mark <= 100) {// an 'A' grade could be A+, A or A-
  if (mark > 90)
    cout << "You get grade A+." << endl;
  else if (mark > 80)
    cout << "You get grade A." << endl;
  else
    cout << "You get grade A-." << endl;
}
else if (…)
  :
```

Example Nested `if`

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

```
if (mark >= 70 && mark <= 100) {// an 'A' grade could be A+, A or A-
  if (mark > 90)
    cout << "You get grade A+." << endl;
  else if (mark > 80)
    cout << "You get grade A." << endl;
  else
    cout << "You get grade A-." << endl;
}
else if (…)
```

Example Nested `if`

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

```
if (mark >= 70 && mark <= 100) {// an 'A' grade could be A+, A or A
  if (mark > 90) {
    cout << "You get grade A+." << endl;
    if (mark == 100)
      cout << "Full mark!" << endl;
  }
  else if (mark > 80)
    cout << "You get grade A." << endl;
  else
    cout << "You get grade A-." << endl;
}
else if …
```

Do Not Mix Up == and =

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

Bad practice!

Typical typo error!

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 **1**/TRUE) in C++, "**true**"
  - FALSE is represented by **0**, in C++, "**false**"
  - Non-zero represents TRUE, in C++, "**true**"

C++ Syntax is Different from the Math Syntax

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

Can we express the above condition as follows?

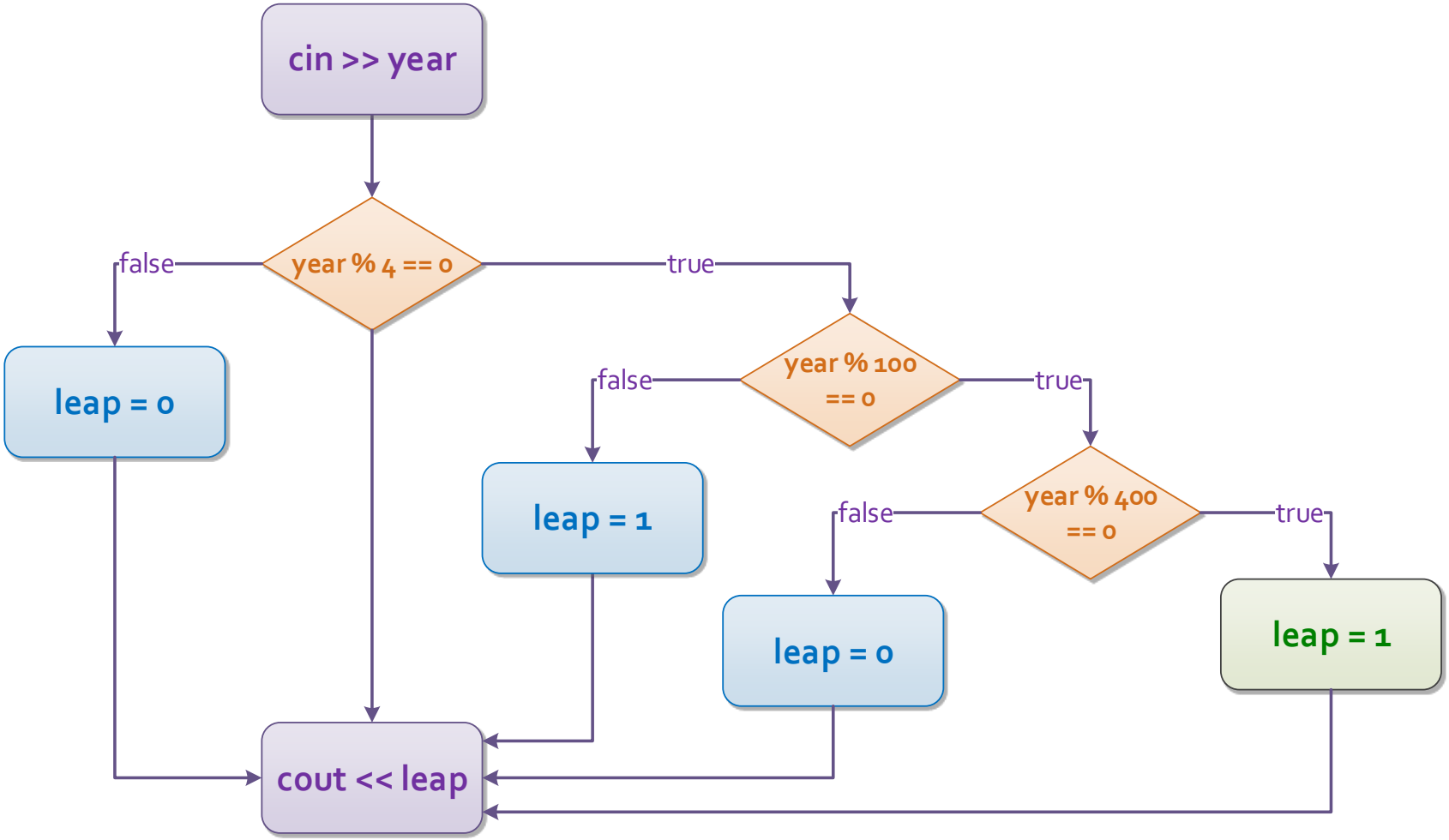
```
if (70 <= mark <= 100)
    ......
```

Ans: **NO**

Note. PASS does NOT accept!

- && ✗ and
- || ✗ or
- ! ✗ not

Example: Check if a year is a leap year



Example: Check if a year is a leap year

in C++

```
//  check leap year : use nested if .. else
unsigned int year;
cout << "Please input year: ";
cin >> year;

if (year % 4 == 0)
    if (year % 100 == 0)
        if (year % 400 == 0)
            cout << year << " is a leap year." << endl;
        else
            cout << year << " is not a leap year." << endl;
    else
        cout << year << " is a leap year." << endl;
else
    cout << year << " is not a leap year." << endl;
```

Short-circuit Evaluation

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

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

Short-circuit Evaluation

- Evaluation of expressions containing **&&** and **||** stops as soon as the outcome **`true`** or **`false`** is known.
  - This is called ***short-circuit evaluation***
  - E.g., if (`1<0 && y=2`) is the same as if (`false && true`), and thus **`y=2` is not executed**!

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

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

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***
  - *E.g.,* `if(1<0 && y==2)` *is the same as if (false && true/false), and thus …*
- Thus, Short-circuit evaluation improves program *efficiency*
- Short-circuit evaluation exists in some other programming languages too, e.g., C and Java

Short-circuit Evaluation

- Given integer variables `i`, `j` and k, what are the outputs when running the program below?

x && y

```
k = (i=2) && (j=2);
cout << i << j << endl;
k = (i=0) && (j=3);
cout << i << j << endl;
```

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

x || y

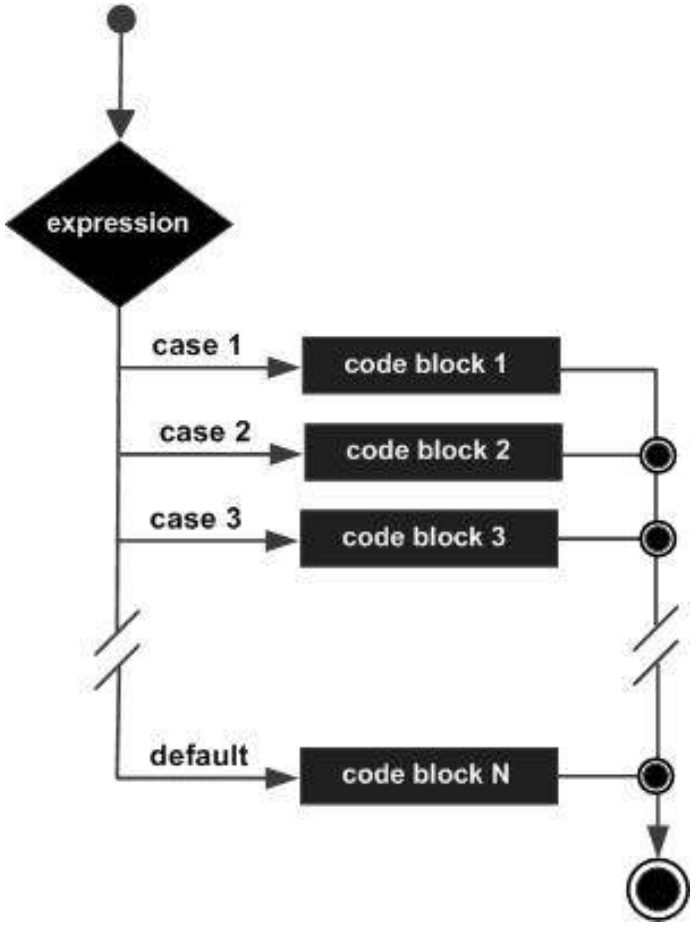
```
k = i || (j=4);
cout << i << j << endl;
k = (i=2) || (j=5);
cout << i << j << endl;
```

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

`switch` statement: Syntax

```
switch (expression) { //similar to if(expression)


}
```



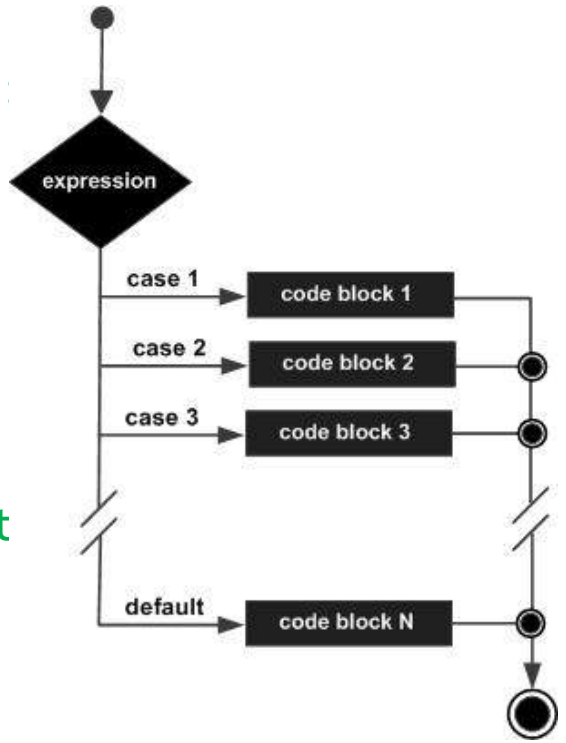
switch Statement

```
switch (expression) {
  case constant_expr1 : statement1
  case constant_expr2 : statement2
  ...
  ...
  case constant_exprN : statementN
  default : statement
 }
```

switch Statement

```
switch (expression) { //e.g., switch(x)
case constant_expression1://e.g. case 1:
    statement(s);
    break;          //optional
// you can have any number of case statement
default : //optional
    statement(s);
}
```

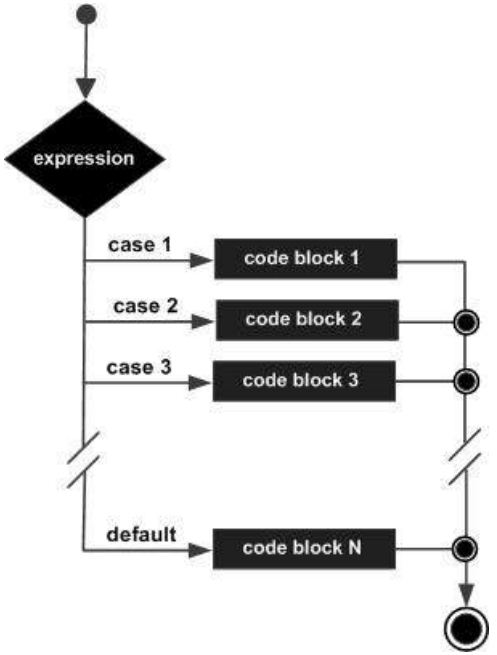
Terminate the switch when a break statement is encountered



- Go to the case label having a 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

`switch` Statement

```
switch (expression) { // similar to if (expression)
case constant_expression1:   // case 1:
    statement(s);
    break;          //optional
case constant_expression2:   // case 2:
    statement(s);
    break;          //optional
// you can have any number of case statements.
default :           //optional
    statement(s);
}
```



- If there is ***no break*** statement, execution "*falls through*" to the next statement in the succeeding case

Example switch

```
int x;
cin >> x;
switch (x) {
   case 0:
     cout << "Zero";
     break; // no braces is needed
   case 1:
     cout << "One";
     break;
   case 2:
     cout << "Two";
     break;
   default:
     cout << "Greater than two";
}  //end switch
```

Example switch

```
int x;
cin >> x;
switch (x) {
   case 0:
     cout << "Zero";
     x = 1;
     break; // no braces is needed
   case 1:
     cout << "One";
     break;
   case 2:
     cout << "Two";
     break;
   default:
     cout << "Greater than two";
}  //end switch
```

`switch` Statement

- Semantics
  - Evaluate the ***`switch` expression*** which results in an ***integer*** type (**int, long, short, char**)
  - Go to the ***`case`*** label having a ***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 successful case

Ternary Conditional Operator (`?:`)

- Syntax:
  - `expr1 ? expr2 : expr3`
  - example. `a > b ? a = 1 : a = 0`
- Semantics
  - `expr1` is evaluated
  - If the result is `true`, execute `expr2` ; else `expr3` is executed
  - The value of the whole expression `?:` is the value of expression evaluated at the end
- Example
  - find the larger of `x` and `y`

```
int larger = (x > y) ? x : y;
```

NOT recommended

Summary

- Boolean logic has two values only; **`true`** or **`false`**.
- Conditional statements are the statements that only execute under certain conditions.
- There are two types of conditional statements

```
if (…) {…} else {…}
switch(…) { case : break}
```