CS2311 Computer Programming

LT03: Control Flow - Condition

Computer Science, City University of Hong Kong

Semester B 2022-23

Lab Updates

- Check PASS !
- Deadline of each Lab submission: the Tuesday next week

Quick Review: Data, Operators, and BasicIO

- C++ basic syntax
- Variable and constant
- Operators
- Basic I/O

Quick Review: Tokens in C++

```
#include <iostream>
using namespace std ;
int main ( ) {
    float r , area ;
    cout << “input circle radius ” ;
    cin >> r ;
    area = 3.1415926 * r * r ;
    cout << “area is ” << area << endl ;
    return 0 ;
}
```

preprocessor

keywords

identifiers

operators

string constants

numeric constants

punctuators

Quick Review: Keywords

Data type	char	double	float	int	bool
	long	short	signed	unsigned	void
Flow control	if	else	switch	case	while
	break	default	for	do	continue
Others	using	namespace	true	false	sizeof
	return	const	class	new	delete
	operator	public	protected	private	friend
	this	try	catch	throw	struct
	typedef	enum	union		

Quick Review: Variable and Constant

- Every variable/constant has ***5*** attributes
- *Address*: location of data in memory storage
- *Value*: content in memory storage
- *Name*: identifier of the variable
- *Type*: C++ is a strictly typed language, variables and constants must belong to a data type
  - E.g., numerical, character, logic, other…
- *Scope*: it defines the region within a program where the variable/constant can be accessed, and also the *conflict domain*

Quick Review: Variable Declaration

- Variable and constants must be declared before use
- Variable declaration format

  data_type variable_identifier ;

- *Optionally*, you can set the initial value of variable during declaration
- Examples

```
int age ;
float bathroom_temperature = 28, bedroom_temperature = 30 ;
char initial ;
char student_name[20] ;
```

Quick Review: Variable Declaration - example

- In some cases, when not setting the initial value of variable during declaration

```
int sum ;
sum += 10;
cout << sum << endl;
```

Quick Review: Type Conversion

- *Implicit* type conversion
  - *Binary expression*: lower-ranked operand is promoted to higher-ranked operand, e.g.,

```
int r = 2;
double pi = 3.14159;
cout << pi * r * r << "\n";
```

  - *Assignment*: right operand is promoted/demoted to match the variable type on the left, e.g.,

```
double a = 1.23;
int b = a;
```

9. long double
8. double
7. float

6. long long
5. long
4. int
3. short

2. char

1. bool

Quick Review: Type Conversion

- *Explicit* type conversion (type-casting)

```
int a = 3;
double b = (double)a;
```

- *Demoted values might change or become invalid*

```
double a = 3.1;
int b = (int)a;
cout << b << endl;
```

```
double a = 3.9;
int b = (int)a;
cout << b << endl;
```

9. long double
8. double
7. float

6. long long
5. long
4. int
3. short

2. char

1. bool

Quick Review: Operators

- An operator specifies an operation to be performed on some values
  - These values are called the operands of the operator

- Some examples: +, -, *, /, %, ++, --, >>, <<

- Some of these have meanings that depend on the context
  - e.g., << means different operations in

```
cout << a << endl;
int b = a << 1;
```

Quick Review: Precedence & Associativity of Operators

Operator Precedence (high to low)	Associativity
`::`	None
`.` `->` `[]`	Left to right
`()` `++(postfix)` `--(postfix)`	Left to right
`+(n)` `-(n)` `++(prefix)` `--(prefix)`	Right to left
`*` `/` `%`	Left to right
`+` `-`	Left to right
`=` `+=` `-=` `*=` `/=` `etc.`	Right to left

Example I:
```
a=b+++c
a=(b++)+c;
```

Example II:
```
int a, b=1;
a=b=3+1;
```

Quick Review: Increment & Decrement Operators

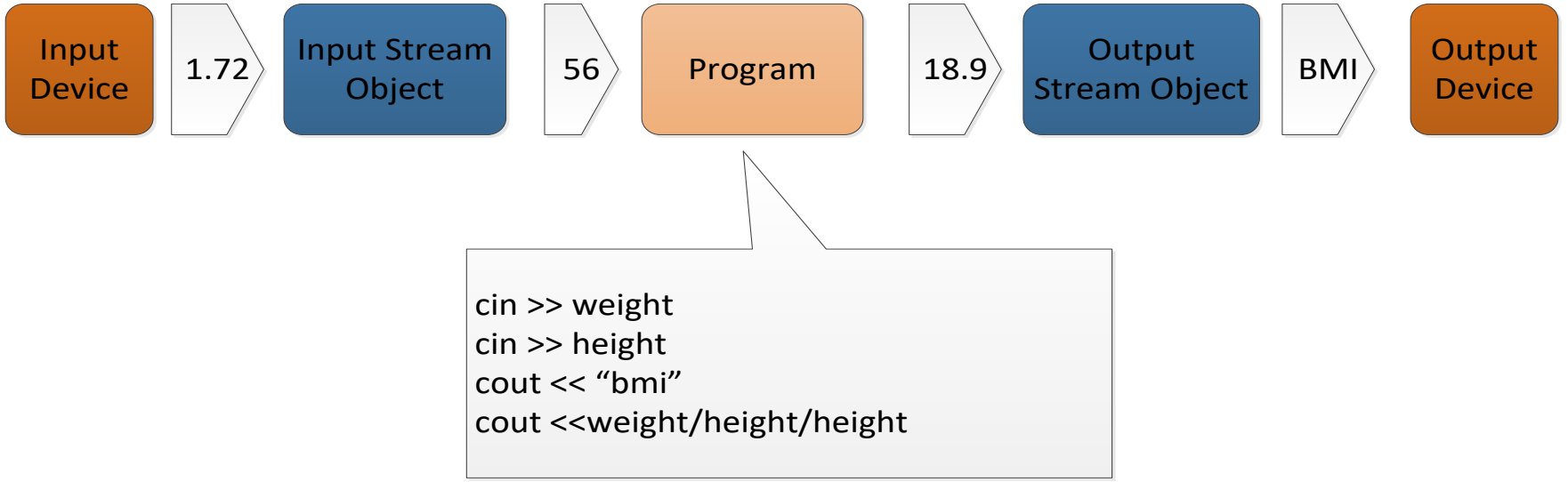
- Increment and decrement operators: `++` and `--`
  - `k++` and `++k` is equivalent to `k=k+1;`
  - `k--` and `--k` is equivalent to `k=k-1;`
- Post-increment and post-decrement: `k++` and `k--`
  - k's value is altered **AFTER** the expression is evaluated

  e.g., `a = k++` is equivalent to (1) `a = k`, (2) `k = k+1`
- Pre-increment and pre-decrement: `++k` and `--k`
  - k's value is altered **BEFORE** the expression is evaluated

  e.g., `a = ++k` is equivalent to (1) `k = k+1`, (2) `a = k`

Quick Review: Basic I/O(cin and cout)

- C++ comes with an iostream package (library) for basic I/O
- cin and cout are objects defined in iostream for keyboard input and screen display, respectively
- To read data from cin and write data to cout, we need to use input operator **(>>)** and output operator **(<<)**



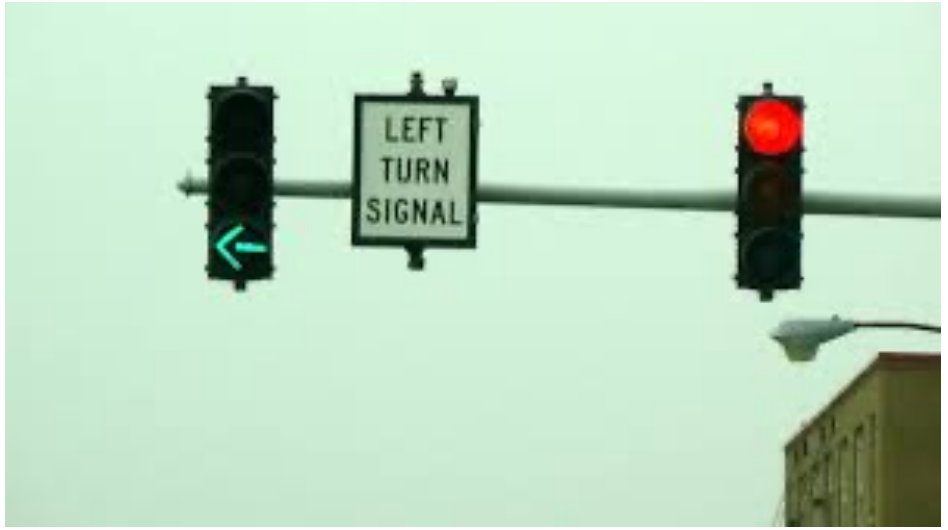
Quick Review: What values are printed?

```
...
int a = 1, b = 0;
b = 1.8+(a++);
cout << b << endl;
cout << a << endl;

b +=++a;
cout << setprecision(1); // means output n significant digits
cout << scientific << (double)b<< endl;
cout << a << endl;
...
```

Today: Conditional Statements

- We make decisions everyday
  - AC-1? 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



Today's Outline

- Logical data type, operators and expressions
- If statement
  - Simple
  - Nested
- Switch statement

Logical Data Type: *bool*

- Takes only two values: *true* and *false*
- Numeric values are *1* (true) and *0* (false)
- the *lowest-ranked* data type
- Length: 1 byte

```
bool x = false, y = true;
cout << sizeof(bool) << endl; // 1
cout << x << " " << y << endl; // 0 1
cout << x + 6 << " " << y + 3.14; // 6 4.14
```

9. long double
8. double
7. float

6. long long
5. long
4. int
3. short

2. char

1. bool

Logical Data Type: *bool*

- when a *higher-ranked* type is casted to *bool*, only 0 is converted to false, all non-zero values are converted to true

```
bool x = 0, y = 3.14, z = 0x1100;
cout << x << " " << y << " " << z << endl; // 0 1 1
```

9. long double
8. double
7. float

6. long long
5. long
4. int
3. short

2. char

1. bool

Logical Data Type: *bool*

- when a *higher-ranked* type is casted to *bool*, only 0 is converted to false, all non-zero values are converted to true

```
bool x = 0, y = 3.14, z = 0x1100;
cout << x << " " << y << " " << z << endl; // 0 1 1
```

- different from demoted conversion of numeric types, which is *direct cut*

```
short a = 0xab0011;
cout << a; // 17
```

9. long double
8. double
7. float

6. long long
5. long
4. int
3. short

2. char

1. bool

Comparative Operators

- Binary operators which accept two operands and compare them

relational operators	syntax	example
Less than	`<`	`x < y`
Greater than	`>`	`z > 1`
Not greater than	`<=`	`b <= 1`
Not less than	`>=`	`c >= 2`

equality operators	syntax	example
Equal to	`==`	`a == b`
Not equal to	`!=`	`B != 3`

Logical Operators: AND (&&) and OR (||)

- Used for *combining* two logical values and *create* a new logical values
- Logical AND (&&)
  - return true if both operands are true
  - otherwise return false
  - example: 18 < age && age < 60
- Logical OR (||)
  - return false if both operands are false
  - otherwise return true

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

Logical Operator: NOT (!)

- Logical-NOT (!) is a *unary* operator that *takes one operand* and *inverts its value*

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

x	!x
true	false
false	true

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`

Logical Expressions

- Expressions that take comparative or logical operators
  - `x == 3`
  - `y == x`
  - `x >= 2`
  - `x != y`

- The value of a logical expression is bool, i.e., can be true or false only

DO NOT MIX: x=y VS x==y

```
int x = 0, y = 4, z = 8;
cout << x=y << endl;
```

```
cout << y==z << endl;
```

DO NOT MIX: x=y VS x==y

```
int x = 0, y = 4, z = 8;

cout << x=y << endl;    // This line will print 4, because:
                        // x=y is an assignment expression!
                        // It copies the value of y to x.
                        // The value of an assignment expression equals to
                        // the value of the right operand,
                        // i.e., 4 in this example

cout << y==z << endl;
```

DO NOT MIX: x=y VS x==y

```
int x = 0, y = 4, z = 8;

cout << x=y << endl;    // This line will print 4, because:
                        // x=y is an assignment expression!
                        // It copies the value of y to x.
                        // The value of an assignment expression equals to
                        // the value of the right operand,
                        // i.e., 4 in this example

cout << y==z << endl;   // This line will print 0 because:
                        // y==z is a logical expression!
                        // and y doesn't equal to z
```

DO NOT MIX: a<x<b VS a<x && x<b

```
double a = 0.5;

cout << 0<a && a<1 << endl;



cout << 0<a<1 << endl;
```

DO NOT MIX: a<x<b VS a<x && x<b

```
double a = 0.5;

cout << 0<a && a<1 << endl; // will print 1, because:
                            // the value of 0 < a is true (0<0.5)
                            // the value of a < 1 is true (0.5<1)
                            // the operator && combines the two values
                            // and creates a new value
                            // which is true (printed as 1) in this example

cout << 0<a<1 << endl;
```

DO NOT MIX: a<x<b VS a<x && x<b

```
double a = 0.5;

cout << 0<a && a<1 << endl;  // will print 1, because:
                             // the value of 0 < a is true (0<0.5)
                             // the value of a < 1 is true (0.5<1)
                             // the operator && combines the two values
                             // and creates a new value
                             // which is true (printed as 1) in this example

cout << 0<a<1 << endl;       // will print 0, because:
                             // 0 < a < 1 is equivalent to (0 < a) < 1
                             // in this example, it's equivalent to (0<0.5) < 1
                             // i.e., true < 1, which equals to false
                             // and printed as 0
```

Short-circuit evaluation

- Short-circuit evaluation can improve program efficiency
- Short-circuit evaluation exists in some other programming languages, e.g., C and Java
- example:

```
int x = 0;
bool b = (x == 0 || ++x == 1);
// b equals true; x equals 0

b = (x != 0 && ++x == 1);
// b equals false; x equals 0
```

Short Circuit Evaluation

- Evaluation of logical expressions containing && and || stops as soon as the outcome *true* or *false* is known

For &&: the value of x&&y is `false` as long as x is false

in this case, the value of y doesn't matter and *is NOT evaluated*

For ||: the value of x||y is `true` as long as x is true

in this case, the value of y doesn't matter and *is NOT evaluated*

Short Circuit Evaluation

- Example I:

```
int x = 0, y = 2;
bool a = (x==0 || y==1);  // we know that a must equal to true after
                          // evaluating x==0 (which is true), it doesn't
                          // matter if y equals to 1 or not

bool b = (x!=0 && y==2);  // we know that b must equals to false after
                          // evaluating x!=0  (which is false), it doesn't
                          // matter if y equals to 2 or not
```

Short Circuit Evaluation

- Example II:

```
int a = 0, b = 0;
bool x = (a==0 || b=1);
cout << b << endl;
cout << x << endl;


bool y = (a==0 && b=1);
cout << b << endl;
cout << y << endl;
```

Short Circuit Evaluation

- Example II:

```
int a = 0, b = 0;

bool x = (a==0 || b=1);    // we know that x must equal to true after
cout << b << endl;         // evaluating a==0 (which is true)
cout << x << endl;         // in this case b=1 is not evaluated
                           // therefore, the value of b is still 0

bool y = (a==0 && b=1);
cout << b << endl;
cout << y << endl;
```

Short Circuit Evaluation

- Example II:

```
int a = 0, b = 0;

bool x = (a==0 || b=1);    // we know that x must equal to true after
cout << b << endl;         // evaluating a==0 (which is true)
cout << x << endl;         // in this case b=1 is not evaluated
                           // therefore, the value of b is still 0

bool y = (a==0 && b=1);
cout << b << endl;         // what value will be printed and why?
cout << y << endl;
```

Short Circuit Evaluation

- Example III:

```
int a = 0, b = 0;

bool x = (a!=0 && b=1);
cout << b << endl;          // what value will be printed and why?
cout << x << endl;


bool y = (a!=0 || b=1);
cout << b << endl;          // what value will be printed and why?
cout << y << endl;
```

Quick Summary

- Comparative operators
  - less than <
  - not less than <=
  - greater than >
  - not greater than >=
  - equal to ==
  - not equal to !=
- Logical operators
  - logical AND &&
  - logical OR &&
  - watch out to SHORT CIRCUIT!
  - logical NOT !
- The value of a logical expression is bool
  - i.,e, true or false

Exercise

- Determine which of the logical expressions have a value `true`, assuming that the value of the variable *`count`* is 0 and the value of the variable *`limit`* is 10

a. `(count == 0) && (limit < 20)`

b. `count == 0 && limit < 20`

c. `(limit > 20) || (count < 5)`

d. `!(count == 12)`

e. `(count == 1) && (x < y)`

f. `(count < 10) || (x < y)`

g. `!( ((count < 10) || (x < y)) && (count >= 0) )`

h. `((limit / count) > 7) || (limit < 20)`

i. `(limit < 20) || ((limit / count) > 7)`

j. `((limit / count) > 7) && (limit < 0)`

k. `(limit < 0) && ((limit / count) > 7)`

l. `(5 && 7) + (!6)`

* credit: Savitch, Walter J. Absolute C++. Pearson Education, 2006.

Today's Outline

- Logical data type, operators and expressions
- If statement
  - Simple
  - Nested
- Switch statement

Conditional Statements

- In decision making process, logical value can be used to determine the actions to take

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

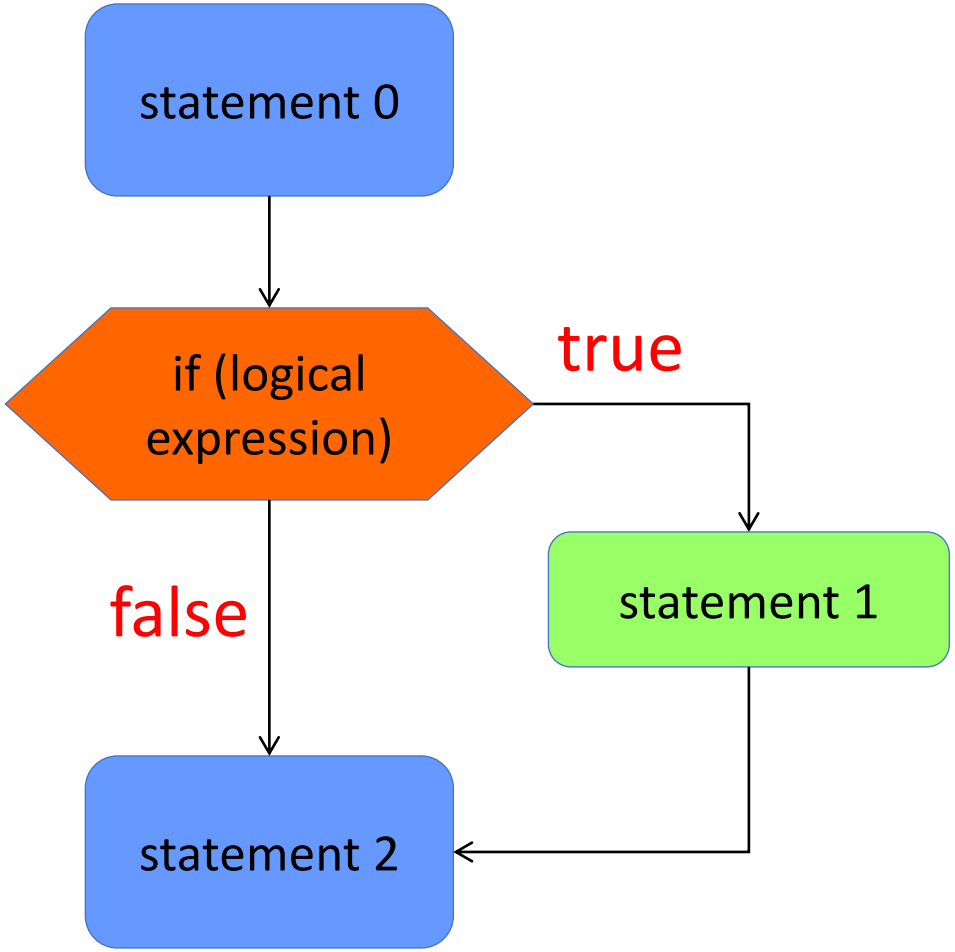
- In programming, certain statements will only be executed when certain condition is fulfilled. We call them *conditional statements*

if Statement: Basic Syntax

```
statement 0;
if (logical expression)
    statement 1;
statement 2;
```

- statement 1 will be executed if logical expression is evaluated to true, for example

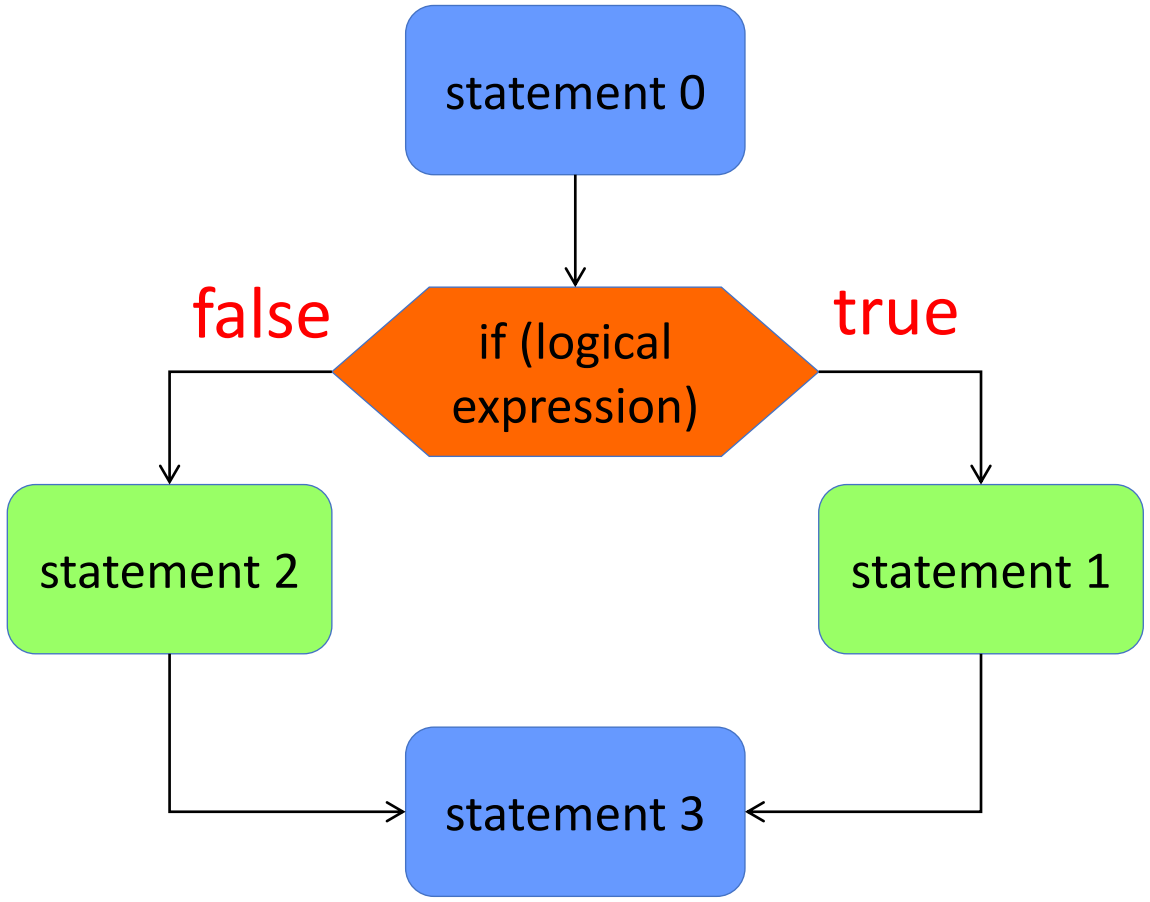
```
cin >> x;
if (x < 0)
    x = -x;
cout << x;
```



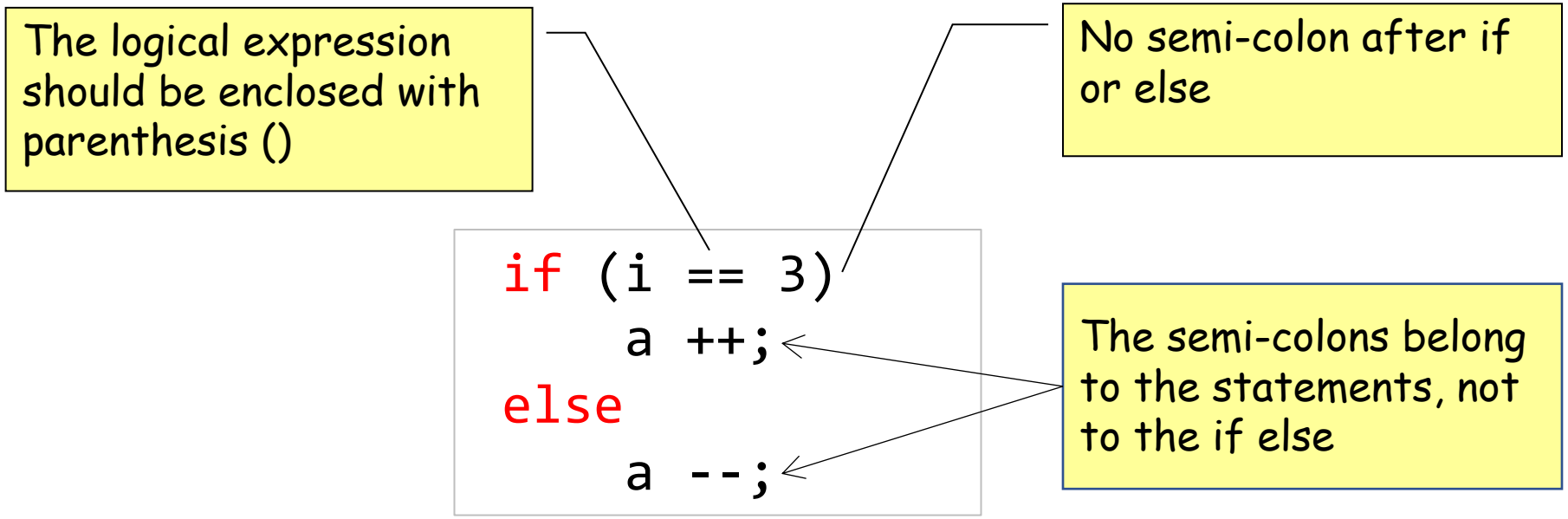
if Statement: Two-way Condition

```
statement 0;
if (logical expression)
    statement 1;
else
    statement 2;
statement 3;
```

- if logical expression is true, statement 1 will be executed
- If logical expression is false, statement 2 will be exected



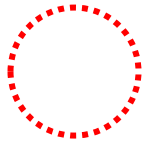
if Statement: Some Syntax Notes



if Statement: Some Syntax Notes

- Watch out to *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 program on the right, x=3 statement will NOT be executed if x equals to 5.
- For program on the left, x=3 statement will be always executed.

if Statement: Inline Ternary

- Also known as *inline* if-then-else constructs

- Syntax
  - expr1 ? expr2 : expr3 ;

- Semantics
  - expr1 is evaluated as the condition
  - if the value of expr1 is non-zero/true, then execute expr2;
  - else execute expr3

```
int a, b, c;
cin >> a;
cin >> b;
a>=b ? c=a : c=b;
cout << c;
```

if Statement: Inline Ternary

- The value of the whole inline ternary expression equals to the expression evaluated *at the end*
- For example

```
int a, b, c;
cin >> a;
cin >> b;
c = a>=b ? a : b;
cout << c;
```

Precedence & Associativity

Operator precedence (high to low)	Associativity
::	none
() ++ (post) -- (post)	Left to right
~ ! ++ (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**

Precedence & Associativity

	Expression	Fully-Parenthesized Expression
1	a + b + c	((a + b) + c)
2	a = b = c	(a = (b = c))
3	c = a + b	(c = (a + b))
4	a + b * c / d % - g	(a + (((b * c) / d) % (-g)))
5	++i++	(++(i++))
6	a += b += c += d	(a += (b += (c += d)))
7	d = a && !b || c	(d = ((a && (!b)) || c))
8	z = a == b ? ++c : --d	(z = ((a == b) ? (++c) : (--d)))

Compound if

- Group multiple statements into one block using {} to be executed for one branch

```
if (logical expression) {
      statement 1;
      …
      statement n;
} else {
      statement n+1;
      …
      statement n+m;
}
```

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 if it were a **single statement**

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

Compound if: Example 1

```
int mark;
cout << "What is your exam mark?\n";
cin >> mark;
if (mark >= 30)
    cout << "You passed the exam of CS2311!\n";
```

- If the input mark is greater than or equal to 34, the yellow statement is executed.

Compound if: Example 2

```
int mark;
cout << "What is your exam mark?\n";
cin >> mark;
if (mark >= 30) {
    cout << "You passed the exam of CS2311!\n";
    cout << “Congratulations!\n”;
} else
    cout << “You failed CS2311 … \n”;
```

- If more than 1 statements are specified within an if branch, group the statements in a pair of braces { }
- The else statement is executed when the `mark>=30` is false

Compound if: Example 3

```
int mark;
cout << "What is your exam mark?\n";
cin >> mark;
if (mark >= 30) {
    cout << "You passed the exam of CS2311!\n";
    cout << “Congratulations!\n”;
} else
    cout << “You failed CS2311 … \n”;
    cout << “You need to retake CS2311.\n”;
```

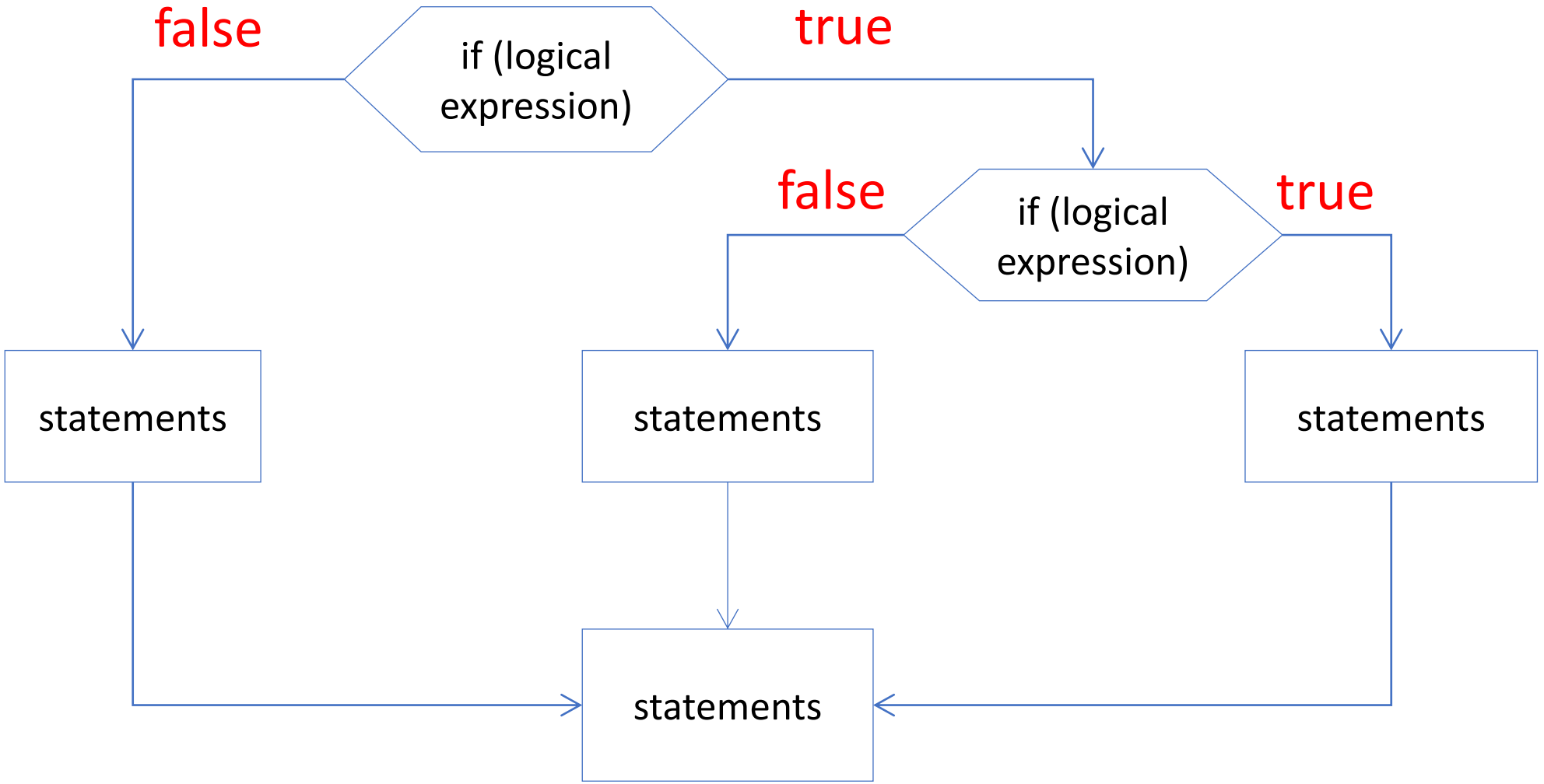
- Suppose the user inputs 40, what will be printed and why?

Compound if: Example 3

```
int mark;
cout << "What is your exam mark?\n";
cin >> mark;
if (mark >= 30) {
    cout << "You passed the exam of CS2311!\n";
    cout << “Congratulations!\n”;
} else {
    cout << “You failed CS2311 … \n”;
    cout << “You need to retake CS2311.\n”;
}
```

- Don’t forget to use { } to enclose the statements in the else branch

Beyond Two-way Condition



Multi-way Condition: Construct

- In C++, multi-way condition can be constructed as,

```
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 {
   statements when both logical expression 1 and 2 are false
}
```

Multi-way Condition: An Example

- Input a mark, display grade according to
- A: [90, 100], B: [75, 90), C: [55, 75), D: [0, 55)

Multi-way Condition: An Example

- Input a mark, display grade according to
- A: [90, 100], B: [75, 90), C: [55, 75), D: [0, 55)

```
if (mark < 0 || mark > 100)
   cout << “invalid mark \n”;
if (mark >= 90 && mark <= 100)
   grade = ‘A’;
if (mark < 90 && mark >= 75)
   grade = ‘B’;
if (mark < 75 && mark >= 55)
   grade = ‘C’;
if (mark < 55 && mark >= 0)
   grade = ‘D’;
```

Multi-way Condition: An Example

- Input a mark, display grade according to
- A: [90, 100], B: [75, 90), C: [55, 75), D: [0, 55)

```
if (mark < 0 || mark > 100)
   cout << “invalid mark \n”;
if (mark >= 90 && mark <= 100)
   grade = ‘A’;
if (mark < 90 && mark >= 75)
   grade = ‘B’;
if (mark < 75 && mark >= 55)
   grade = ‘C’;
if (mark < 55 && mark >= 0)
   grade = ‘D’;
```

```
if (mark < 0 || mark > 100)
   cout << “invalid mark \n”;
else if (mark >= 90)
   grade = ‘A’;
else if (mark >= 75)
   grade = ‘B’;
else if (mark >= 55)
   grade = ‘C’;
else
   grade = ‘D’;
```

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

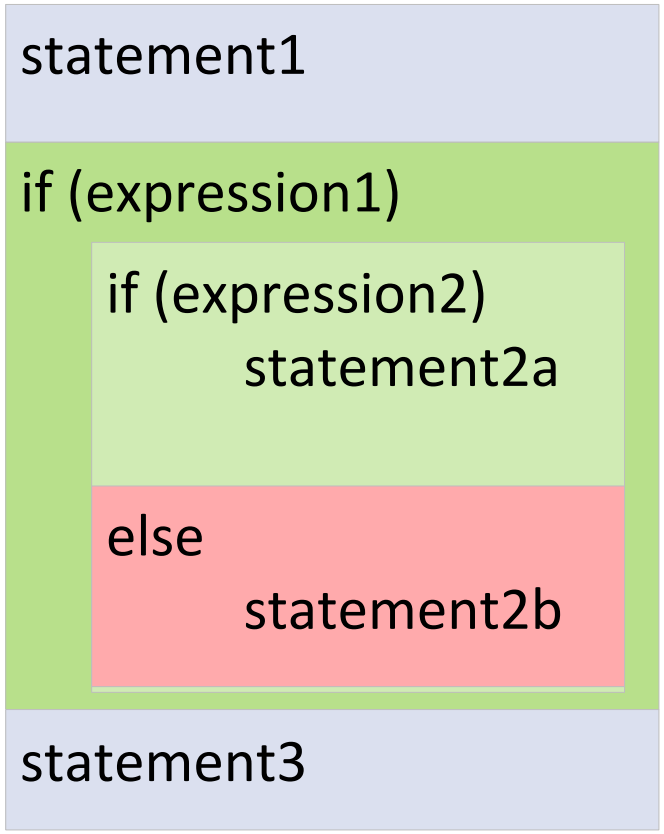
Can we express the above condition as follows?

```
if (70<=mark<=100)
     ……
```

Ans: No

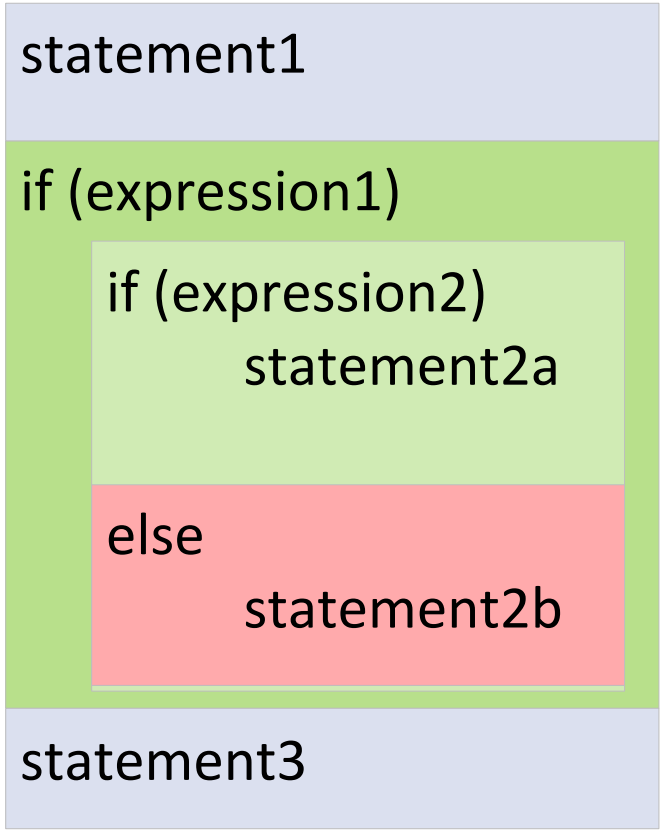
Nested if

- An if-else statement is included within another if or else statement



Nested if

- An if-else statement is included within another if or else statement




```
if (mark>=90 && mark<=100) {
  // divide A into can be A-, A or A+
  if (mark>97)
      cout << "You get grade A+\n";
  else if (mark>93)
      cout << "You get grade A \n";
  else
      cout << "You get grade A-\n";
}
```

Nested if (cont'd)

- Consider the two indentation formats of the same program

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

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

- With which "if" the else statement is associated?

Nested if (cont'd)

- Consider the two indentation formats of the same program

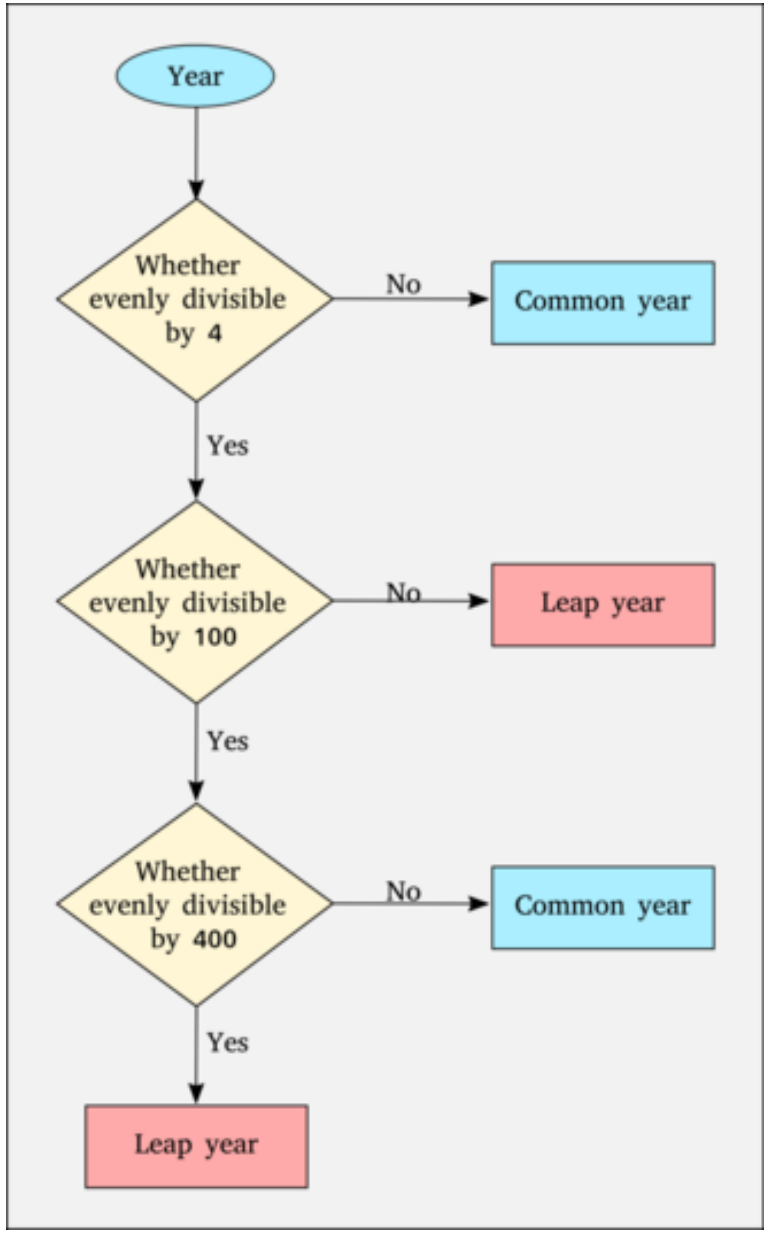
```
if (a==1)
    if (b==2)
        cout << "***\n";
else
    cout << "###\n";
```

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

- With which "if" the else statement is associated?
- An else is attached to the nearest if

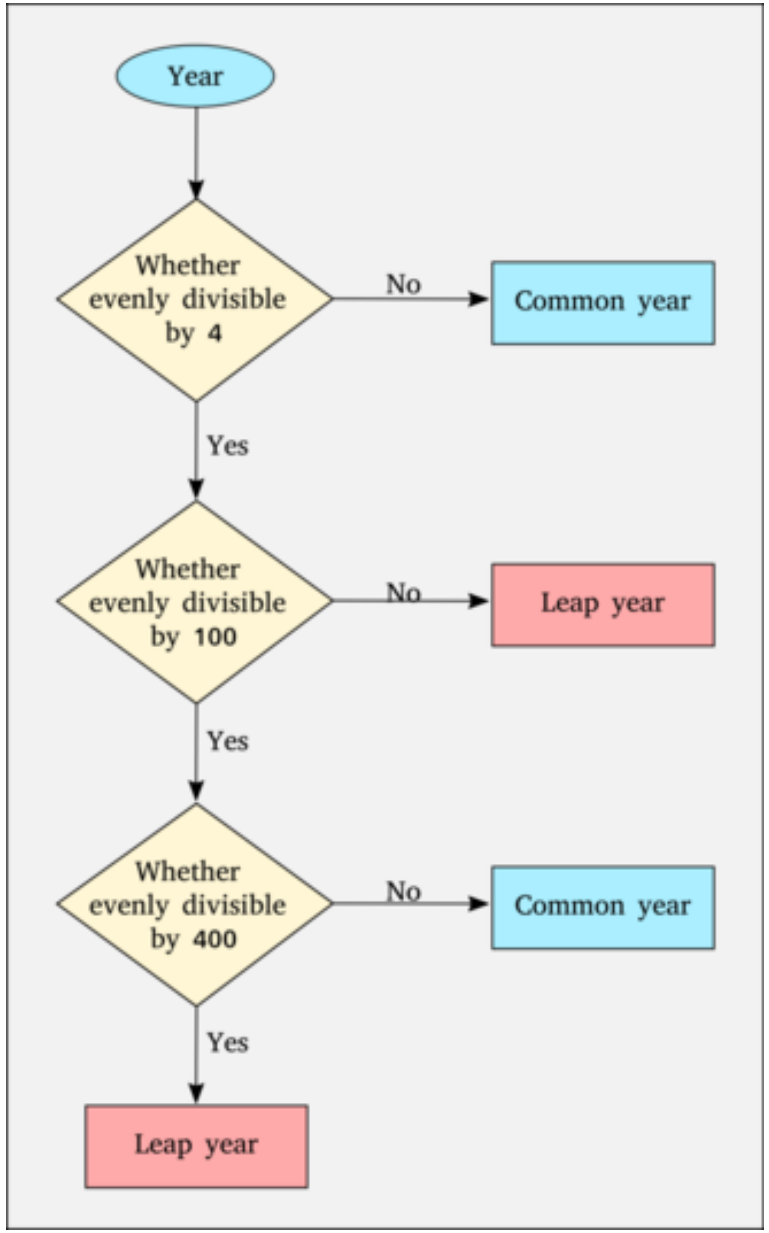
Nested if: An Example

- Check if a year is leap year
- A leap year is a calendar year that contains an additional day added to February to keep the calendar year synchronized with the astronomical year or seasonal year
- These extra days occur in each year that is an integer multiple of 4, except for years evenly divisible by 100, but not by 400



Nested if: An Example

```
4   void main() {
5
6       int year;
7       cout << "Please input year: ";
8       cin >> year;
9       if (year % 4 == 0) {
10          if (year % 100 == 0) {
11              if (year % 400 == 0)
12                  cout << "It is a leap year" << endl;
13              else
14                  cout << "It is not a leap year" << endl;
15          }
16          else
17              cout << "It is a leap year" << endl;
18      }
19      else
20          cout << "It is not a leap year" << endl;
21
22  }
```



Today's Outline

- Logical data type, operators and expressions
- If statement
  - Simple
  - Nested
- Switch statement

Motivation

- Is there a better way to organize the following code?

```
int day_of_week;
cin >> day_of_week;
if (day_of_week < 1 || day_of_week )
       cout << “invalid day\n”;
if (day_of_week == 1) cout << “its Monday\n”;
if (day_of_week == 2) cout << “its Tuesday\n”;
if (day_of_week == 3) cout << “its Wednesday\n”;
if (day_of_week == 4) cout << “its Thursday\n”;
if (day_of_week == 5) cout << “its Friday\n”;
if (day_of_week == 6) cout << “its Saturday\n”;
if (day_of_week == 7) cout << “its Sunday\n”;
```

switch: Syntax and Semantic

- Syntax

```
switch (expression) {
    case constant-expr1:
      statements
      break;    // optional
    ...
    case constant-exprN:
       statements
       break;   // optional
    default:    // optional
       statements
       break;   // optional
}
```

- Semantic
  - Evaluate the 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 expression;
  - when a break statement is encountered, terminate the switch
  - If there is no break statement, execution falls through to the next statement
  - if a match is not found, go to the default label;
  - if default label does not exist, terminate the switch

switch: Example 1

```
int day_of_week;
cin >> day_of_week;
switch (day_of_week) {
    case 1:  cout << "Monday\n";    break;
    case 2:  cout << "Tuesday\n";   break;
    case 3:  cout << "Wednesday\n"; break;
    case 4:  cout << "Thursday\n";  break;
    case 5:  cout << "Friday\n";    break;
    case 6:  cout << "Saturday\n";  break;
    case 7:  cout << "Sunday\n";    break;
    default: cout << "Invalid\n";   break;
}  // end switch
```

switch: Example 1

```
// What happens there is no break ??
int day_of_week;
cin >> day_of_week;
switch (day_of_week) {
    case 1:  cout << "Monday\n";
    case 2:  cout << "Tuesday\n";
    case 3:  cout << "Wednesday\n";
    case 4:  cout << "Thursday\n";
    case 5:  cout << "Friday\n";
    case 6:  cout << "Saturday\n";
    case 7:  cout << "Sunday\n";
    default: cout << "Invalid\n";
}  // end switch
```

switch: Example 1

```
// What happens there is no break ??
int day_of_week;
cin >> day_of_week;
switch (day_of_week) {
    case 1:  cout << "Monday\n";
    case 2:  cout << "Tuesday\n";
    case 3:  cout << "Wednesday\n";
    case 4:  cout << "Thursday\n";
    case 5:  cout << "Friday\n";
    case 6:  cout << "Saturday\n";
    case 7:  cout << "Sunday\n";
    default: cout << "Invalid\n";
}  // end switch
```

- Semantic
  - Evaluate the 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 expression;
  - when a break statement is encountered, terminate the switch
  - If there is no break statement, execution falls through to the next statement
  - if a match is not found, go to the default label;
  - if default label does not exist, terminate the switch

switch: Example 2

- What are the outputs if we enter '3'?
- What are the outputs if we enter '*a*'?

```
cin >> c;// 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++;
        break;
    case ' ': case '\n': case '\t':
        white_character_count++;
        break;
    default:
        other_character_count++;
        break;

}

cout << digit_count << endl;
cout << white_character_count << endl;
cout << other_character_count << endl;
```

Summary

- Boolean logic has two values only; true or false.

- Conditional statements are the statements that only execute under certain conditions.

- In C++, there are two approaches to construct conditional statement
  - if (…){…}else{…}
  - switch(…){case:break…}

Exercise 1

- Police captured four suspects, among whom one is a thief
- Consider the following statement of the suspects
  - A: I'm not the thief
  - B: C is the thief
  - C: D is the thief
  - D: C is lying
- If we know that there is only one liar, then who is the thief? Can you determine who is the thief by writing a program?

Exercises 2

Write a program that prompts the user to enter an integer and determines whether it is

a) divisible by 3 and 5
b) divisible by 3 only
c) divisible by 5 only
d) not divisible by 3 or 5

Expected Output:

Example 1	Example 2
Enter an Integer Number: **15** 15 is divisible by 3 and 5.	Enter an Integer Number: **21** 21 is divisible by 3 only.
Example 3	**Example 4**
Enter an Integer Number: **40** 40 is divisible by 5 only.	Enter an Integer Number: **23** 23 is not divisible by 3 or 5.

Exercises 3

- Write a program that reads 3 integer values from the user. The 3 values are interpreted as representing the lengths of the three sides of a triangle. The program prints a message saying whether the triangle is:
  - ***Equilateral*** (all sides equal),
  - ***Isosceles*** (only 2 sides equal),
  - ***Scalene*** (all sides unequal),
  - or ***Impossible*** (can't form a triangle).

  A triangle can be formed only if the sum of the length of *any 2* sides is greater than the length of the 3rd side and the length of all the sides of the triangle are *positive*.

Expected Output:

Example 1	Example 2
Enter the value of A, B and C: 3 4 5 Scalene	Enter the value of A, B and C: 3 3 3 Equilateral
Example 3	**Example 4**
Enter the value of A, B and C: 5 5 2 Isosceles	Enter the value of A, B and C: 1 2 10 Impossible
Example 5	**Example 6**
Enter the value of A, B and C: 0 2 10 Impossible	Enter the value of A, B and C: 1 -2 10 Impossible