C++ Basics
Part Four

Sai-Keung Wong

National Yang Ming Chiao Tung University

Hsinchu, Taiwan

Intended Learning Outcomes

- Describe the processes of
  - sequence structures,
  - selection structures,
  - repetition structures,
  - short-circuits
- Describe the fall-through mechanism and *break* in a switch block

if and if-else structures

Work flows

Bool type and comparison operators

A and B are variables of the same type.

How to compare two values A and B?

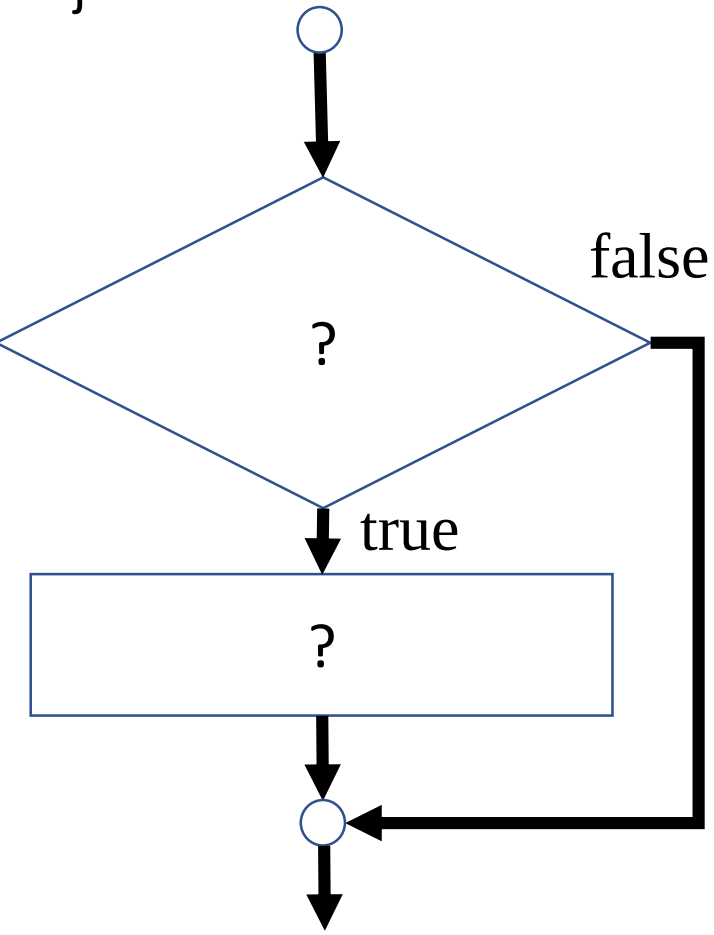
1) Is A greater than B?
2) Is A equal to B?
3) Is A not equal to B?
4) Is A greater than or equal to B?
5) Is the sum of A and B smaller than 2000?

Relational Operators

1) A > B
2) A == B
3) A !=B
4) A >= B
5) A+B < 20000

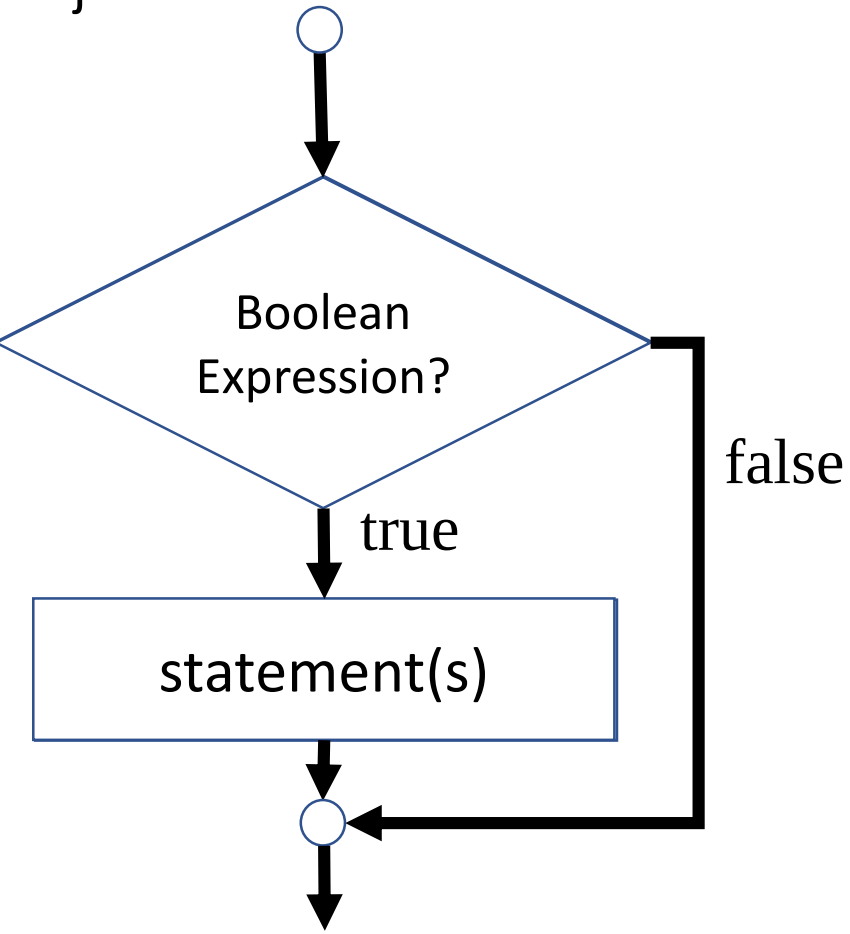
Flowchart: One-way if Statements

```
if (booleanExpression)
{
  statement(s);
}
```



Flowchart: One-way if Statements

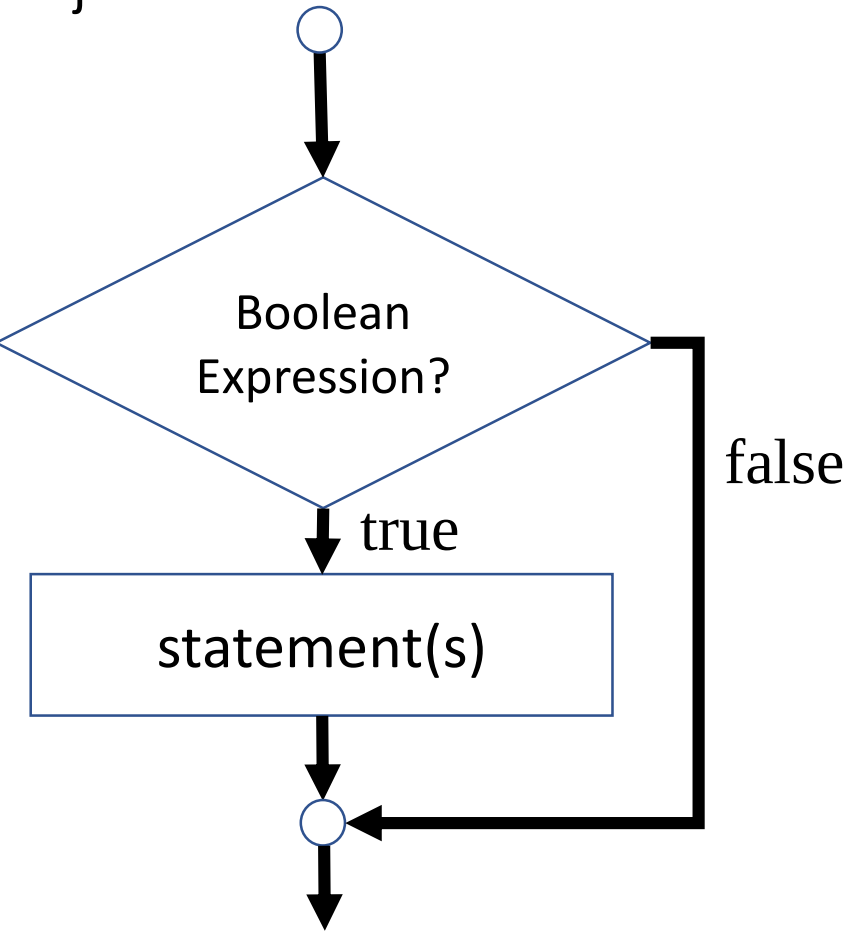
```
if (booleanExpression)
{
  statement(s);
}
```



Flowchart: One-way if Statements

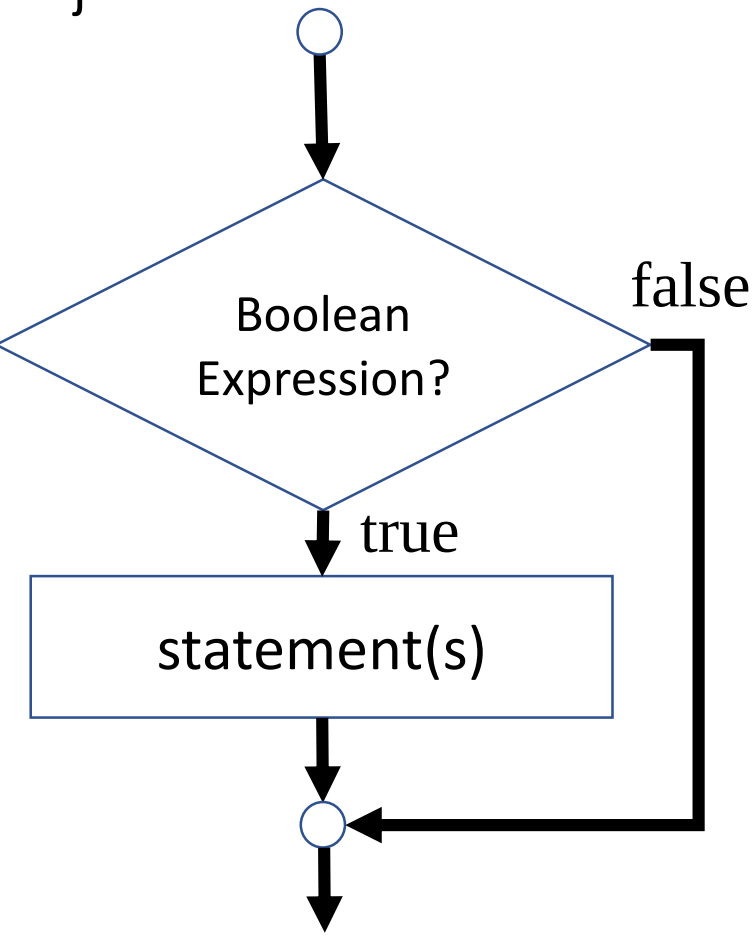
```
if (booleanExpression)
{
  statement(s);
}
```

```
if (relationalExpression)
{
  statement(s);
}
```



Flowchart: One-way if Statements

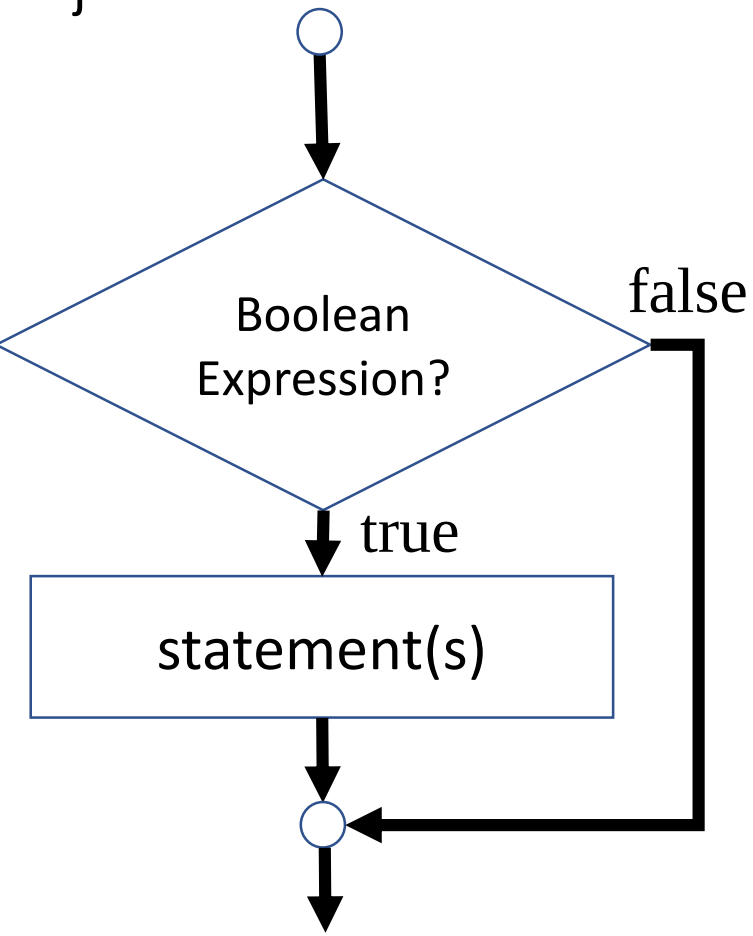
```
if (booleanExpression)
{
  statement(s);
}
```

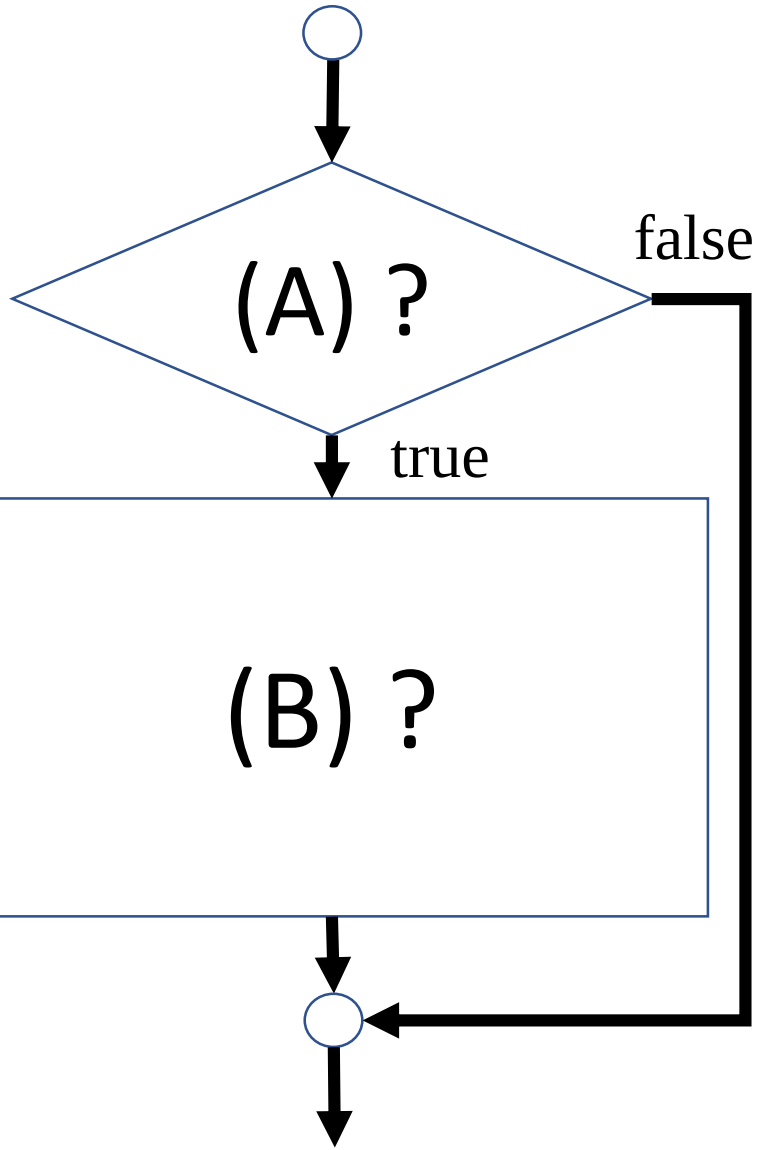



```
if (side >= 0) {
  square_area = side*side;
  cout  << "The area of the square is "
        <<  square_area
        << “.”
        << “Its side length is “ << side << endl;
}
```

One-way if Statements

```
if (booleanExpression)
{
  statement(s);
}
```

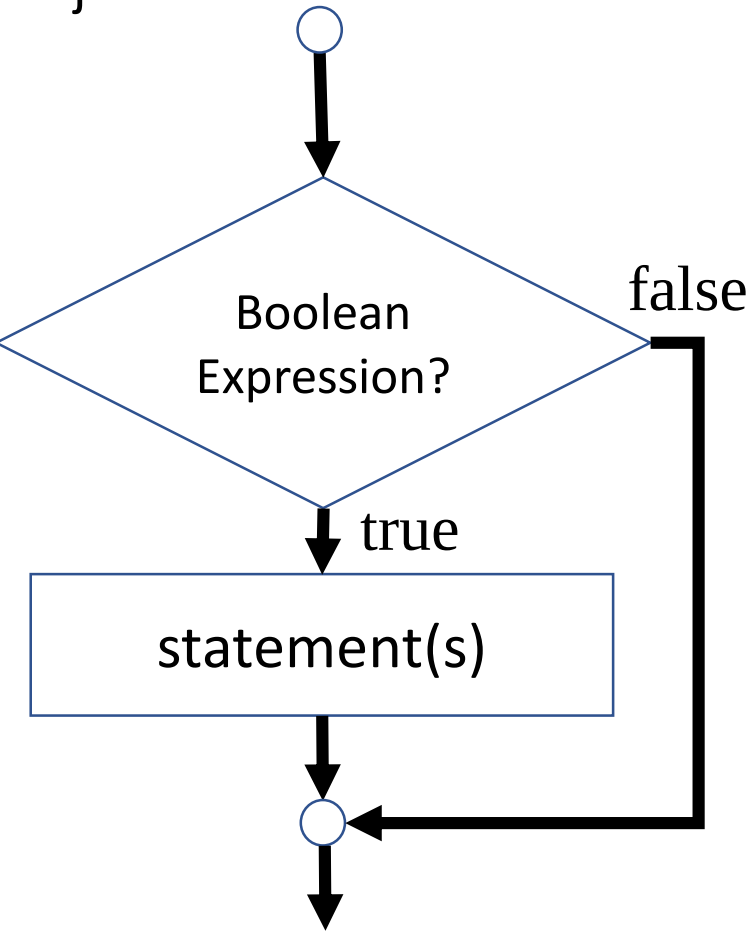


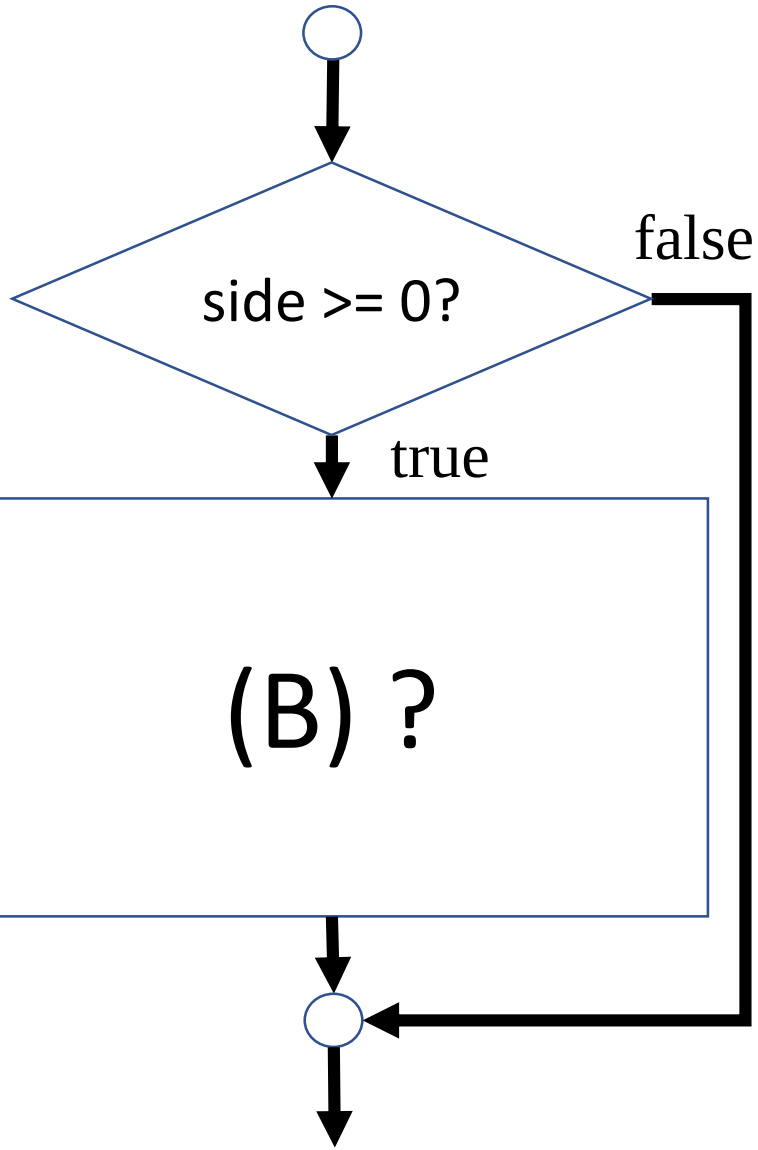



```
if (side >= 0) {
  square_area = side*side;
  cout  << "The area of the square is "
        <<  square_area
        << "."
        << "Its side length is " << side << endl;
}
```

One-way if Statements

```
if (booleanExpression)
{
  statement(s);
}
```

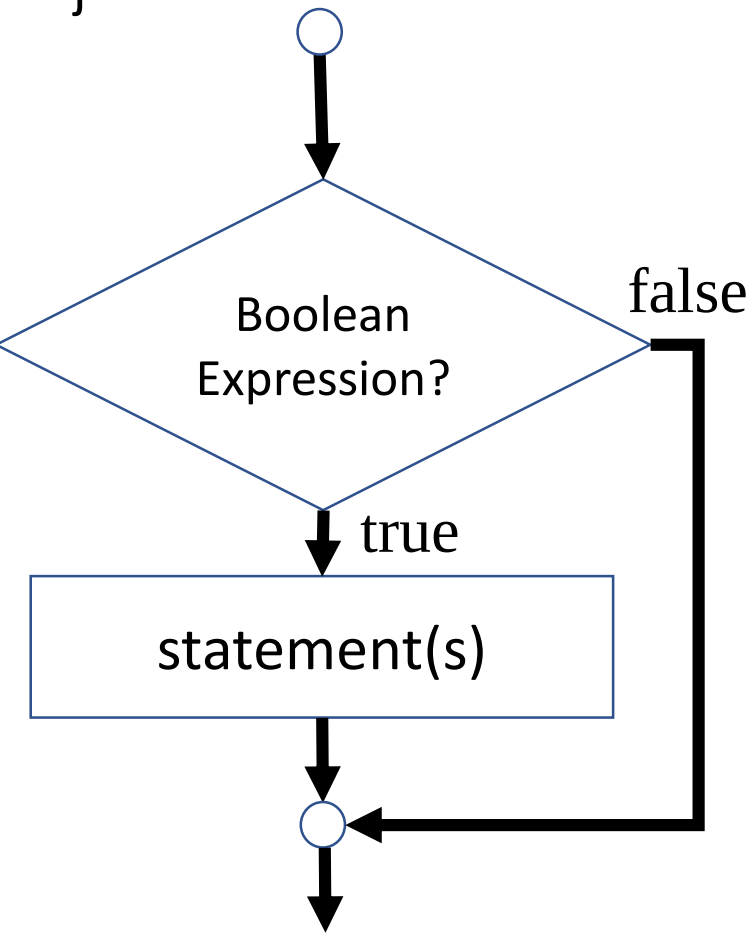


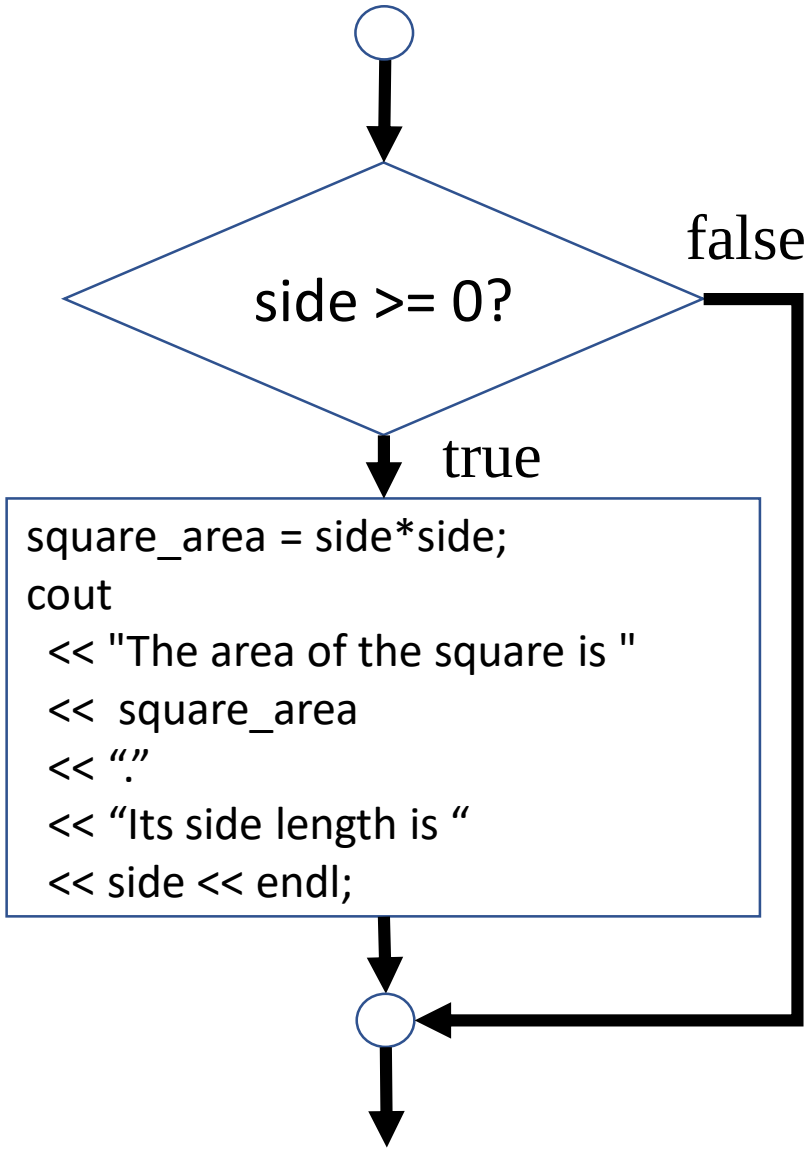



```
if (side >= 0) {
  square_area = side*side;
  cout  << "The area of the square is "
        <<  square_area
        << "."
        << "Its side length is " << side << endl;
}
```

One-way if Statements

```
if (booleanExpression)
{
  statement(s);
}
```






```
if (side >= 0) {
  square_area = side*side;
  cout  << "The area of the square is "
        <<  square_area
        << “.”
        << “Its side length is “ << side << endl;
}
```

Logic error

When we use a semicolon, we must be careful.

An empty statement

```
if (side>=0);
{
   square_area = side*side;
}
```

==
equivalent

```
if (side>=0) { }
{
   square_area = side*side;
}
```

Logic error

It is not a compilation error or a runtime error.

It is a logic error.

It often occurs when the next-line block style is used.

Logic error

When we use a semicolon, we must be careful.

An empty statement

```
if (side>=0);
{
  square_area = side*side;
}
```

==
equivalent

Logic error?

```
if (side>=0) { }
{
  square_area = side*side;
}
```

An empty body

Logic error

It is not a compilation error or a runtime error.

It is a logic error.

It often occurs when the next-line block style is used.

Logic error

When we use a semicolon, we must be careful.

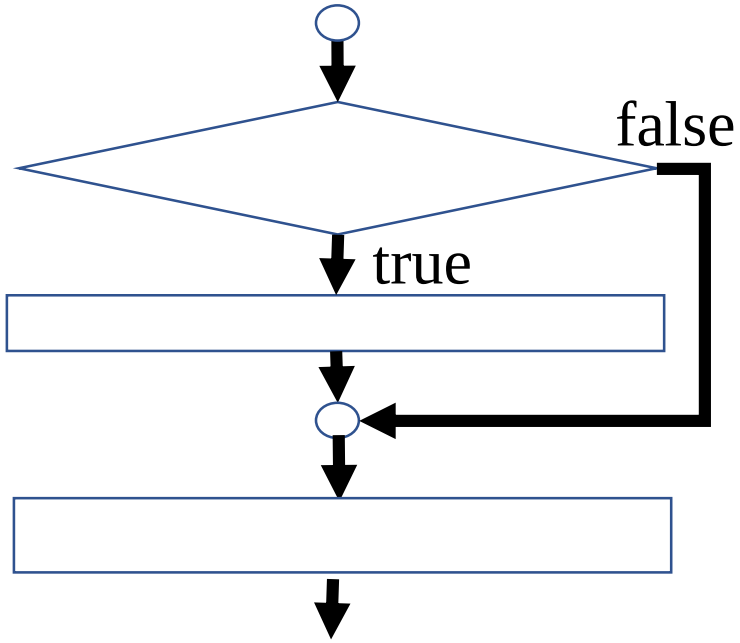
An empty statement

```
if (side>=0);
{
   square_area = side*side;
}
```

==
equivalent

```
if (side>=0) { }
{
   square_area = side*side;
}
```

An empty body



Logic error

When we use a semicolon, we must be careful.

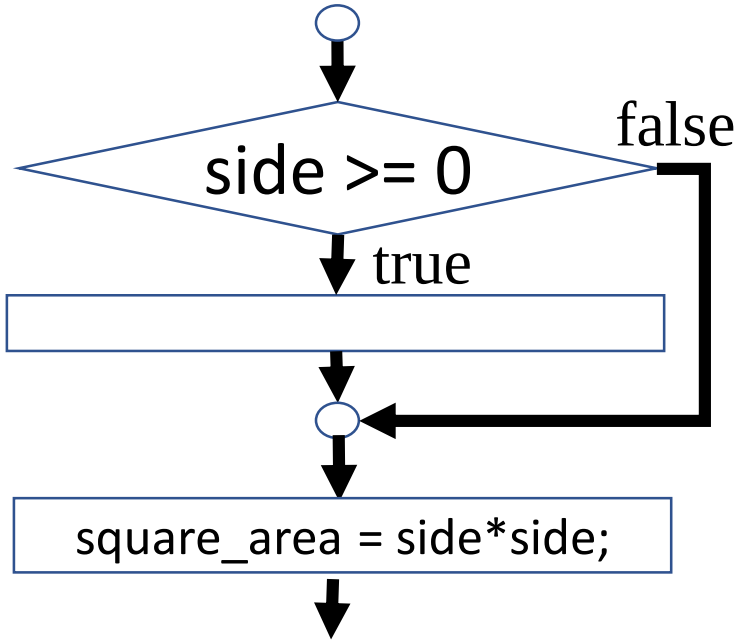
An empty statement

```
if (side>=0);
{
   square_area = side*side;
}
```

==
equivalent

```
if (side>=0) { }
{
   square_area = side*side;
}
```

An empty body



Logic error

When we use a semicolon, we must be careful.

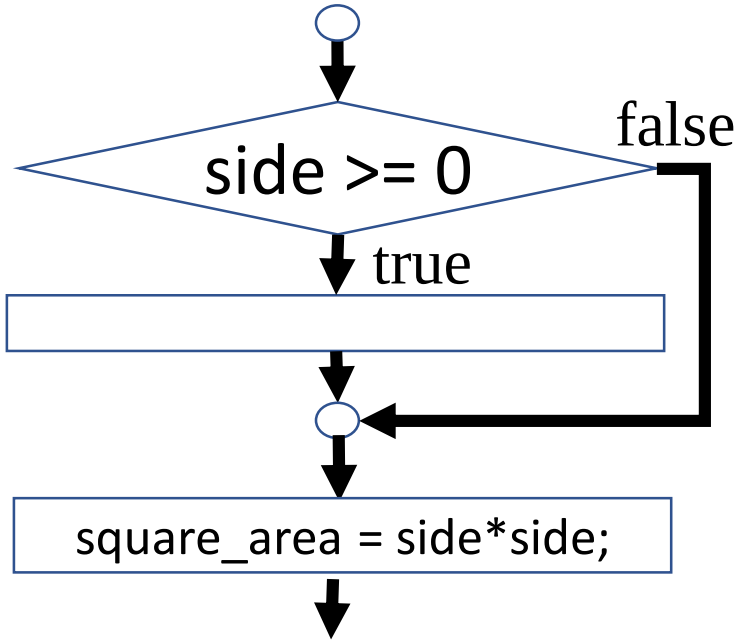
An empty statement

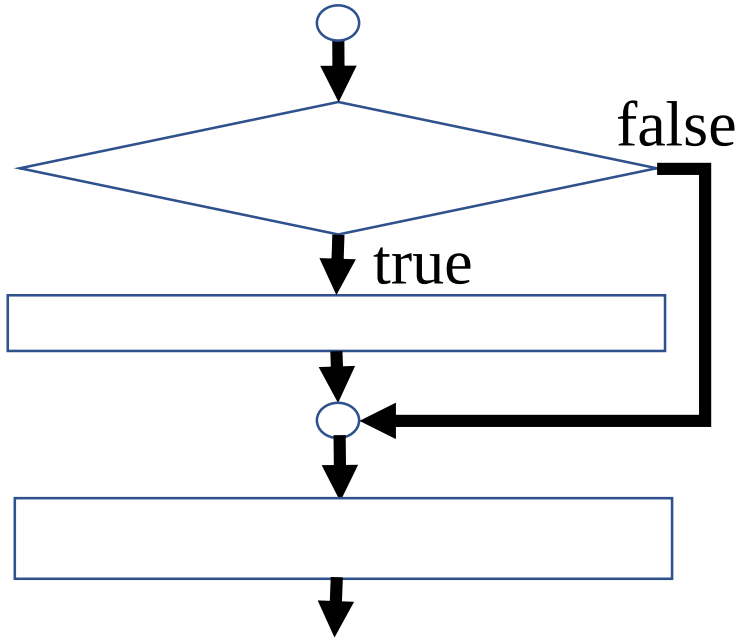
```
if (side>=0);
{
   square_area = side*side;
}
```

==
equivalent

```
if (side>=0) { }
{
   square_area = side*side;
}
```

An empty body





Logic error

When we use a semicolon, we must be careful.

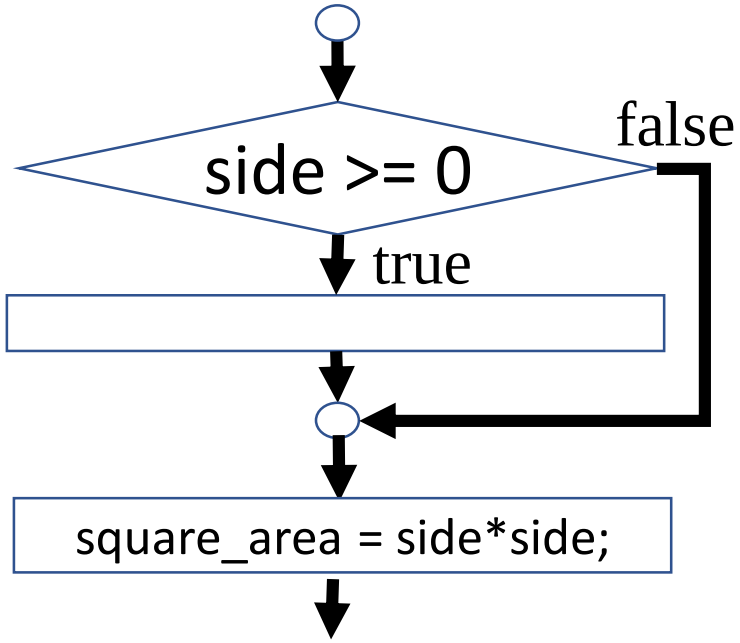
An empty statement

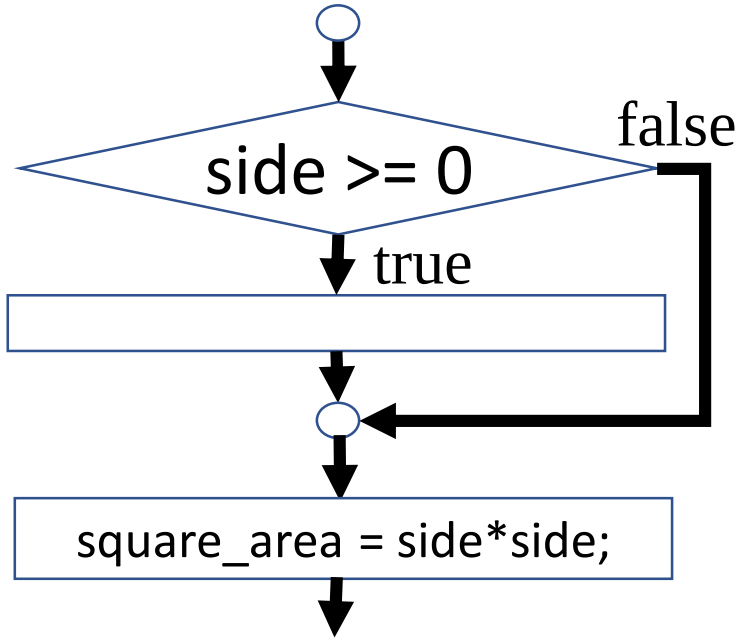
```
if (side>=0);
{
   square_area = side*side;
}
```

==
equivalent

```
if (side>=0) { }
{
   square_area = side*side;
}
```

An empty body





The if-else Statement

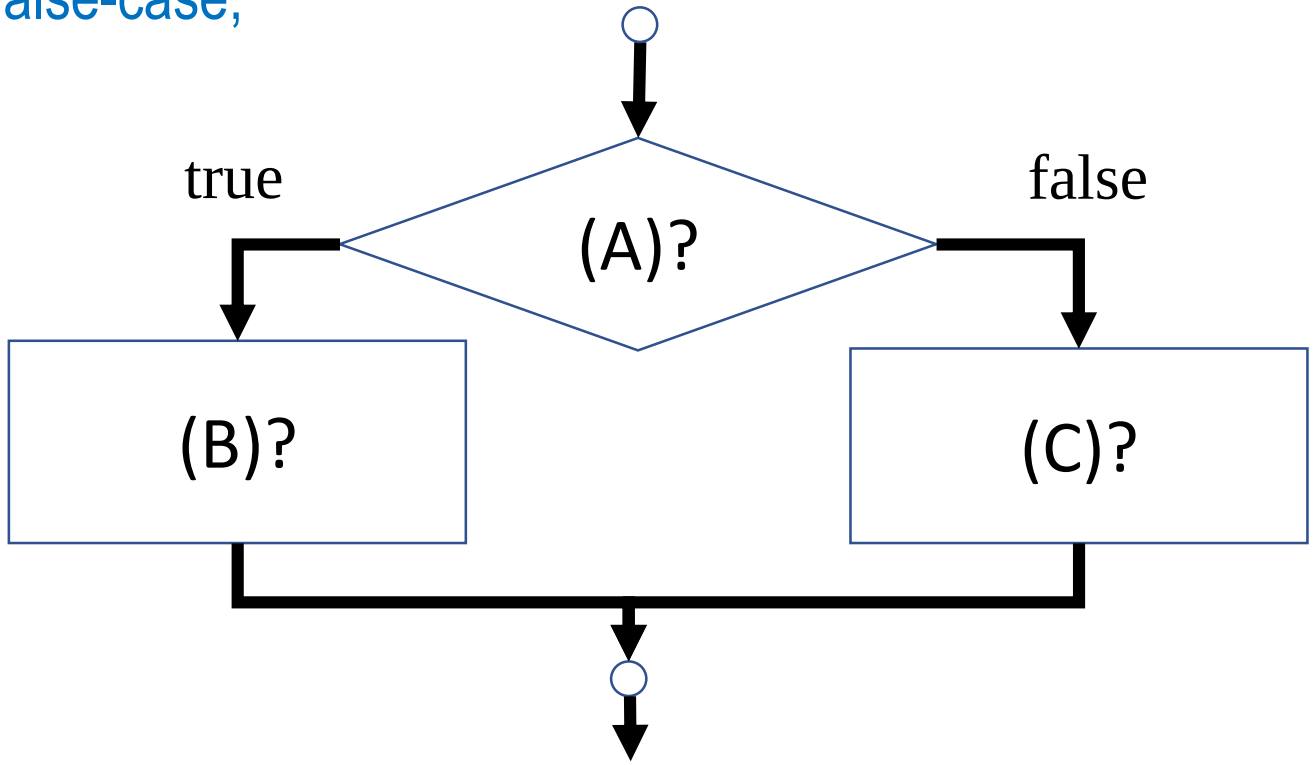
```
if (booleanExpression)
{
  statement(s)-for-the-true-case;
}
else
{
  statement(s)-for-the-false-case;
}
```

The if-else Statement

```
if (booleanExpression)
{
  statement(s)-for-the-true-case;
}
else
{
  statement(s)-for-the-false-case;
}
```

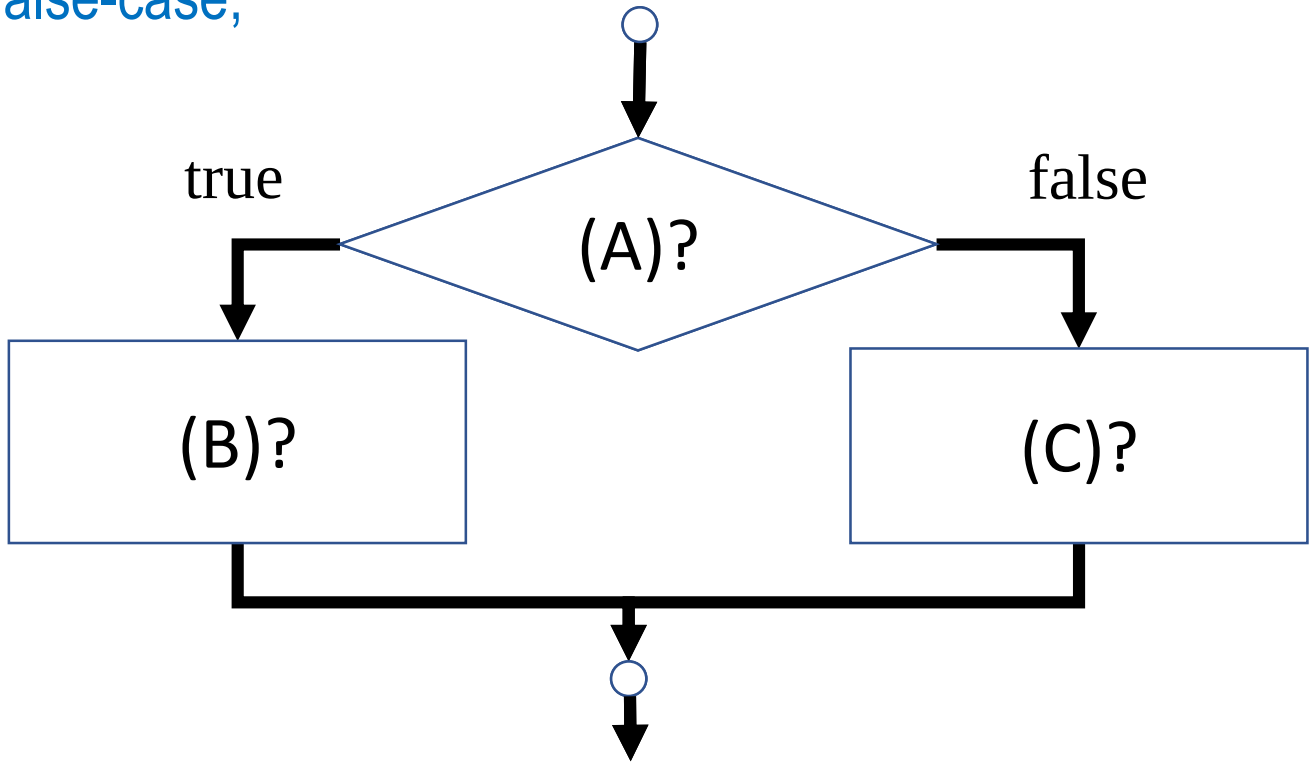
The if-else Statement

```
if (booleanExpression)
{
  statement(s)-for-the-true-case;
}
else
{
  statement(s)-for-the-false-case;
}
```



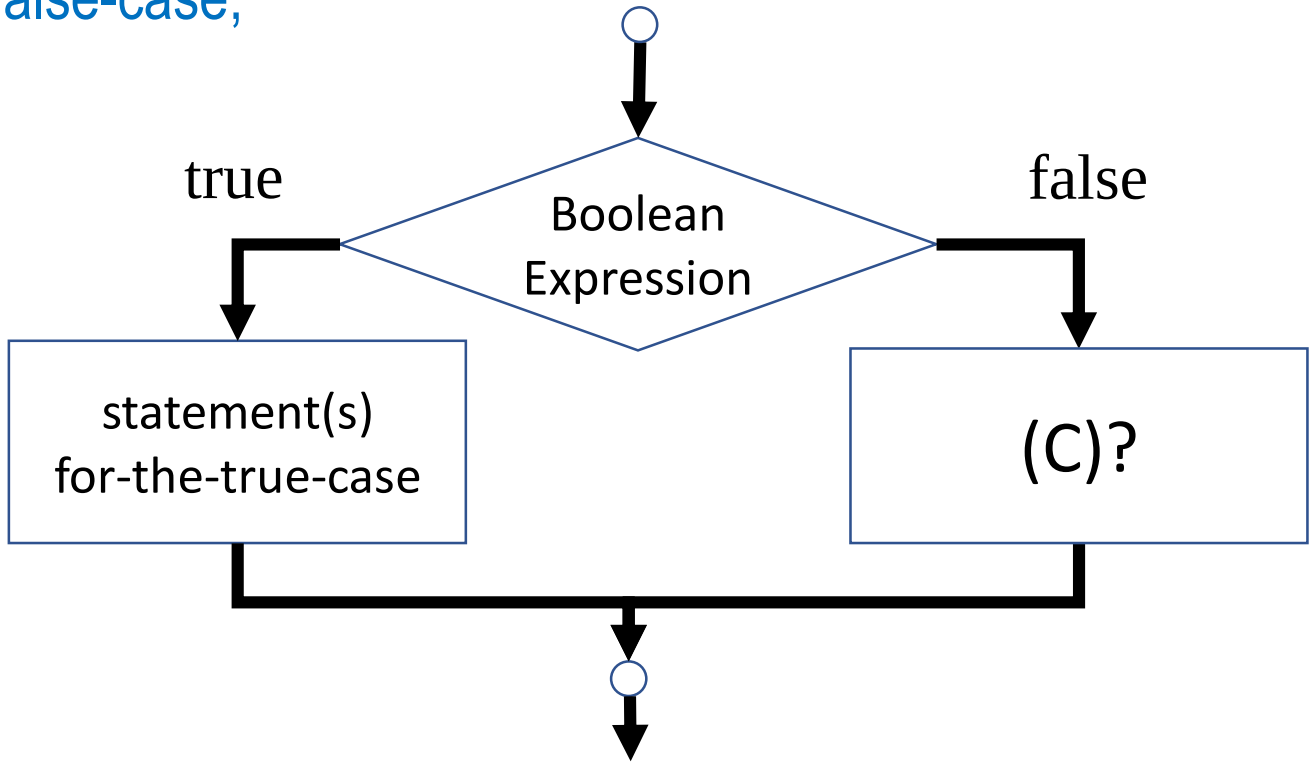
The if-else Statement

```
if (booleanExpression)
{
  statement(s)-for-the-true-case;
}
else
{
  statement(s)-for-the-false-case;
}
```



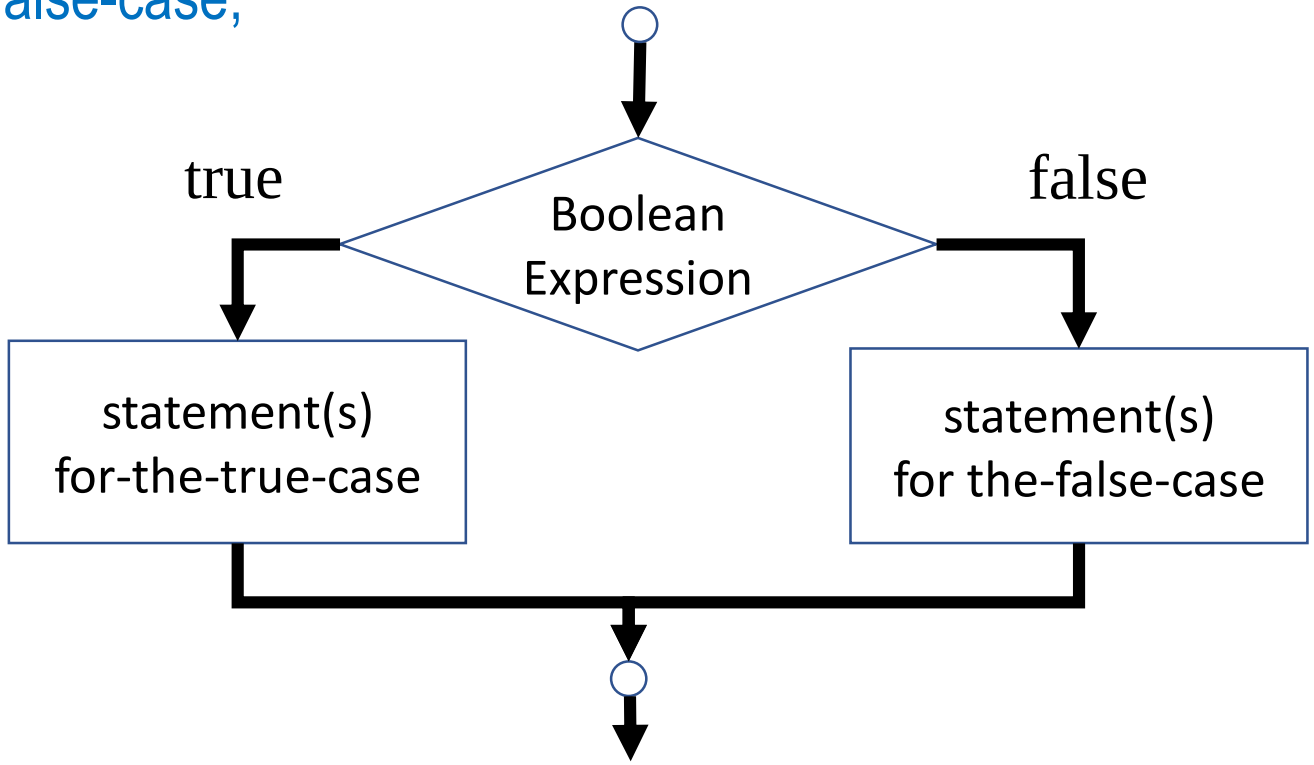
The if-else Statement

```
if (booleanExpression)
{
  statement(s)-for-the-true-case;
}
else
{
  statement(s)-for-the-false-case;
}
```



The if-else Statement

```
if (booleanExpression)
{
  statement(s)-for-the-true-case;
}
else
{
  statement(s)-for-the-false-case;
}
```



Nested if Statements

```
if (a > c)
{
  if (b > c) {
    cout << "a and b are greater than c";
  }
}
```

Nested if Statements

```
if (a > c)
{
  if (b > c) {
    cout << "a and b are greater than c";
  }
}
else {
  cout << "a is less than or equal to c";
}
```

Nested if Statements

```
if (a > c)
{
  if (b > c) {
    cout << "a and b are greater than c";
  }
}
else {
  cout << "a is less than or equal to c";
}
```

==
equivalent

```
if (a > c)
{
  if (b > c)
    cout << "a and b are greater than c";


}
else
  cout << "a is less than or equal to c";
```

Nested if Statements

```
if (a > c)
{
  if (b > c) {
    cout << "a and b are greater than c";
  }
}
else {
  cout << "a is less than or equal to c";
}
```

```
if (a > c)
  if (b > c)
    cout << "a and b are greater than c";
else
  cout << "a is less than or equal to c";
```

Are they the same?

Nested if Statements

```
if (a > c)
{
  if (b > c) {
    cout << "a and b are greater than c";
  }
}
else {
  cout << "a is less than or equal to c";
}
```

```
if (a > c)
  if (b > c)
    cout << "a and b are greater than c";
else
  cout << "a is less than or equal to c";
```

Are they the same?

Nested if Statements

The else-structure is associated with the nearest if-structure.

```
if (a > c)
{
  if (b > c) {
    cout << "a and b are greater than c";
  }
}
else {
  cout << "a is less than or equal to c";
}
```

```
if (a > c)
  if (b > c)
    cout << "a and b are greater than c";
else
  cout << "a is less than or equal to c";
```

Are they the same?

Nested if Statements

The else-structure is associated with the nearest if-structure.

(A)

```
if (a > c)
{
  if (b > c) {
    cout << "a and b are greater than c";
  }
}
else {
  cout << "a is less than or equal to c";
}
```

(B)

```
if (a > c)
  if (b > c)
    cout << "a and b are greater than c";
else
  cout << "a is less than or equal to c";
```

a = 3, b = 2, c = 2;
What are the output for
(A) and (B), respectively?

Nested if Statements

The else-structure is associated with the nearest if-structure.

(A)

```
if (a > c)
{
  if (b > c) {
    cout << "a and b are greater than c";
  }
}
else {
  cout << "a is less than or equal to c";
}
```

```
if (a > c)
  if (b > c)
    cout << "a and b are greater than c";
else
  cout << "a is less than or equal to c";
```

(B)

No output

a = 3, b = 2, c = 2;
What are the output for
(A) and (B), respectively?

a is less than or equal to c

Nested if Statements

The else-structure is associated with the nearest if-structure.

(A)

```
if (a > c)
{
  if (b > c) {
    cout << "a and b are greater than c";
  }
}
else {
  cout << "a is less than or equal to c";
}
```

```
if (a > c)
  if (b > c)
    cout << "a and b are greater than c";
else
  cout << "a is less than or equal to c";
```

(B)

No output

a = 3, b = 2, c = 2;
What are the output for
(A) and (B), respectively?

a is less than or equal to c

Alignment: Multiple Alternative if Statements

```
string grade = "F";
if (score >=90 && score <= 100)
  grade = "A+";
else
  if (score>=85)
     grade = "A";
  else
    if (score>=80)
       grade = "A-";
    else
       if (score >= 77)
          grade = "B+";
```

poor style

```
string grade = "F";
if ( score >=90 && score <= 100 )
  grade = "A+";
else if ( score>=85 )
  grade = "A";
else if ( score>=80 )
  grade = "A-";
else if ( score >= 77)
  grade = "B+";
```

For multiple if-else structures,
align the structures at the left side

Exercise: Trace if-else statement

Suppose score is 78.0

```
string grade = “F”;
if ( score >=90 && score <= 100 )
  grade = “A+”;
else if ( score>=85 )
  grade = “A”;
else if ( score>=80 )
  grade = “A-”;
else if ( score >= 77)
  grade = “B+”;
```

Exercise: Trace if-else statement

Suppose score is 78.0

The condition is false

```
string grade = "F";
if ( score >=90 && score <= 100 )
  grade = "A+";
else if ( score>=85 )
  grade = "A";
else if ( score>=80 )
  grade = "A-";
else if ( score >= 77)
  grade = "B+";
```

Exercise: Trace if-else statement

Suppose score is 78.0

The condition is false

```
string grade = "F";
if ( score >=90 && score <= 100 )
  grade = "A+";
else if ( score>=85 )
  grade = "A";
else if ( score>=80 )
  grade = "A-";
else if ( score >= 77)
  grade = "B+";
```

Exercise: Trace if-else statement

Suppose score is 78.0

The condition is false

```
string grade = "F";
if ( score >=90 && score <= 100 )
  grade = "A+";
else if ( score>=85 )
  grade = "A";
else if ( score>=80 )
  grade = "A-";
else if ( score >= 77)
  grade = "B+";
```

Exercise: Trace if-else statement

Suppose score is 78.0

The condition is **true**

```
string grade = "F";
if ( score >=90 && score <= 100 )
  grade = "A+";
else if ( score>=85 )
  grade = "A";
else if ( score>=80 )
  grade = "A-";
else if ( score >= 77)
  grade = "B+";
```

Exercise: Trace if-else statement

Suppose score is 78.0

Execute the statement

```
string grade = "F";
if ( score >=90 && score <    00 )
  grade = "A+";
else if ( score>=85 )
  grade = "A";
else if ( score>=80 )
  grade = "A-";
else if ( score >= 77)
  grade = "B+";
```

Exercise: Trace if-else statement

Suppose score is 78.0

Exit the if-else structure

```
string grade = "F";
if ( score >=90 && score <    00 )
  grade = "A+";
else if ( score>=85 )
  grade = "A";
else if ( score>=80 )
  grade = "A-";
else if ( score >= 77)
  grade = "B+";
```

Exercise: Trace if-else statement

Suppose score is 99.0

```
string grade = “F”;
if ( score >=90 && score <= 100 )
  grade = “A+”;
else if ( score>=85 )
  grade = “A”;
else if ( score>=80 )
  grade = “A-”;
else if ( score >= 77)
  grade = “B+”;
```

Exercise: Trace if-else statement

Suppose score is 99.0

The condition is **true**

```
string grade = "F";
if ( score >=90 && score <= 100 )
  grade = "A+";
else if ( score>=85 )
  grade = "A";
else if ( score>=80 )
  grade = "A-";
else if ( score >= 77)
  grade = "B+";
```

Exercise: Trace if-else statement

Suppose score is 99.0

Do the assignment

```
string grade = "F";
if ( score >=90 && score <= 100 )
  grade = "A+";
else if ( score>=85 )
  grade = "A";
else if ( score>=80 )
  grade = "A-";
else if ( score >= 77)
  grade = "B+";
```

Exercise: Trace if-else statement

Suppose score is 99.0

Exit the if-else structure

```
string grade = “F”;
if ( score >=90 && score <= 100 )
  grade = “A+”;
else if ( score>=85 )
  grade = “A”;
else if ( score>=80 )
  grade = “A-”;
else if ( score >= 77)
  grade = “B+”;
```

About the else-structure

The else structure matches the nearest if-structure in the same block.

```
int x = 5, y = 4, z = 5;
if ( x > y )
  if (y > z)
     cout << “X” << endl;
else
     cout << “Y” << endl;
```

About the else-structure

The else structure matches the nearest if-structure in the same block.

```
int x = 5, y = 4, z = 5;
if ( x > y )
  if (y > z)
     cout << “X” << endl;
else
     cout << “Y” << endl;
```

==

About the else-structure

The else structure matches the nearest if-structure in the same block.

```
int x = 5, y = 4, z = 5;
if ( x > y )
  if (y > z)
     cout << “X” << endl;
else
     cout << “Y” << endl;
```

==

```
int x = 5, y = 4, z = 5;
if ( x > y )
  if (y > z)
    cout << “X” << endl;
  else
    cout << “Y” << endl;
```

About the else-structure

The else structure matches the nearest if-structure in the same block.

```
int x = 5, y = 4, z = 5;
if ( x > y )
  if (y > z)
      cout << "X" << endl;
else
      cout << "Y" << endl;
```

==

```
int x = 5, y = 4, z = 5;
if ( x > y )
  if (y > z)
    cout << "X" << endl;
  else
    cout << "Y" << endl;
```

!=

```
int x = 5, y = 4, z = 5;
if ( x > y ) {
  if (y > z)
      cout << "X" << endl;
}
else
      cout << "Y" << endl;
```

What is the output of each code fragment?

About the else-structure

The else structure matches the nearest if-structure in the same block.

```
int x = 5, y = 4, z = 5;
if ( x > y )
  if (y > z)
      cout << "X" << endl;
else
      cout << "Y" << endl;
```

x > y is true
y > z is false

Output

Y

```
int x = 5, y = 4, z = 5;
if ( x > y ) {
  if (y > z)
      cout << "X" << endl;
}
else
      cout << "Y" << endl;
```

About the else-structure

The else structure matches the nearest if-structure in the same block.

```
int x = 5, y = 4, z = 5;
if ( x > y )
  if (y > z)
      cout << "X" << endl;
else
      cout << "Y" << endl;
```

x > y is true
y > z is false

Output

Y

```
int x = 5, y = 4, z = 5;
if ( x > y ) {
  if (y > z)
      cout << "X" << endl;
}
else
      cout << "Y" << endl;
```

x > y is true
y > z is false

No output

Simplification of Source code

if and if-else structures

Shorten the source code

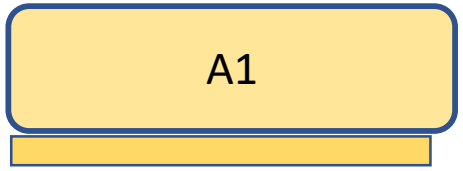
```
bool evenFlg;

if ( number % 2 == 0 )
    evenFlg = true;
else
   evenFlg = false;
```

Shorten the code? Can you?

Shorten the source code

```
bool evenFlg;

if ( number % 2 == 0 )
    evenFlg = true;
else
   evenFlg = false;
```

```
bool evenFlg;

evenFlg =
```



Shorten the source code

```
bool evenFlg;

if ( number % 2 == 0 )
    evenFlg = true;
else
   evenFlg = false;
```

```
bool evenFlg;

evenFlg = number%2 == 0;
```

```
bool evenFlg;

evenFlg = (      A2      );
```

Shorten the source code

```
bool evenFlg;

if ( number % 2 == 0 )
    evenFlg = true;
else
   evenFlg = false;
```

```
bool evenFlg;

evenFlg = number%2 == 0;
```

```
bool evenFlg;

evenFlg = (number%2 == 0);
```

Operator priority issue
The assignment operator = is the A3 in this example.

Shorten the source code

```
bool x;

if ( !( number > 0 ) )
    x = true;
else
   x = false;
```

Shorten the code. Can you?

Shorten the source code

```
bool x;

if ( !( number > 0 ) )
    x = true;
else
   x = false;
```

Shorten the code. Can you?

```
bool x;

x= !( A1 ) ;
```

Shorten the source code

```
bool x;

if ( !( number > 0 ) )
    x = false;
else
   x = true;
```

Shorten the code. Can you?

Shorten the source code

```
bool x;

if ( !( number > 0 ) )
    x = false;
else
   x = true;
```

Shorten the code. Can you?

```
bool x;

x= !!( A1 ) ;
```

Shorten the source code

```
bool x;

if ( !( number > 0 ) )
    x = false;
else
   x = true;
```

Shorten the code. Can you?

```
bool x;

x= !!( number > 0) ;
```

```
bool x;

x= ( A1 ) ;
```

Shorten the source code

```
bool x;

if ( !( number > 0 ) )
    x = false;
else
   x = true;
```

Shorten the code. Can you?

```
bool x;

x= !!( number > 0) ;
```

```
bool x;

x= ( number > 0) ;
```

```
bool x;

x= A1 ;
```

Shorten the source code

```
if ( v% 2 == 0 ) {
    cout “v is even” << endl;
}
else {
    cout << “v is odd” << end;
}
```

```
if (  (A)?  ) {
    cout “v is odd” << endl;
}
else {
    cout << “v is even” << end;
}
```

Shorten the source code

```
if ( v% 2 == 0 ) {
    cout “v is even” << endl;
}
else {
    cout << “v is odd” << end;
}
```

```
if (v % 2) {
    cout “v is odd” << endl;
}
else {
    cout << “v is even” << end;
}
```

A bug
What is the value of a?

```
if ( -3 ) {
        a = 10;
} else {
        a = 0;
}
```

A Code Fragment

```
if ( -3 ) {
    a = 10;
} {
    a = 0;
}
```

B Code Fragment

A bug
What is the value of a?

```
if ( -3 ) {
        a = 10;
} else {
        a = 0;
}
```

A Code Fragment

```
if ( -3 ) {
    a = 10;
} {
    a = 0;
}
```

B Code Fragment

A non-zero value in the Boolean expression is interpreted as true.

A bug
What is the value of a?

(A)

```
if ( -3 ) {
        a = 10;
} else {
        a = 0;
}
```

a = 10

A Code Fragment

```
if ( -3 ) {
    a = 10;
} {
    a = 0;
}
```

(B)

a = 0

B Code Fragment

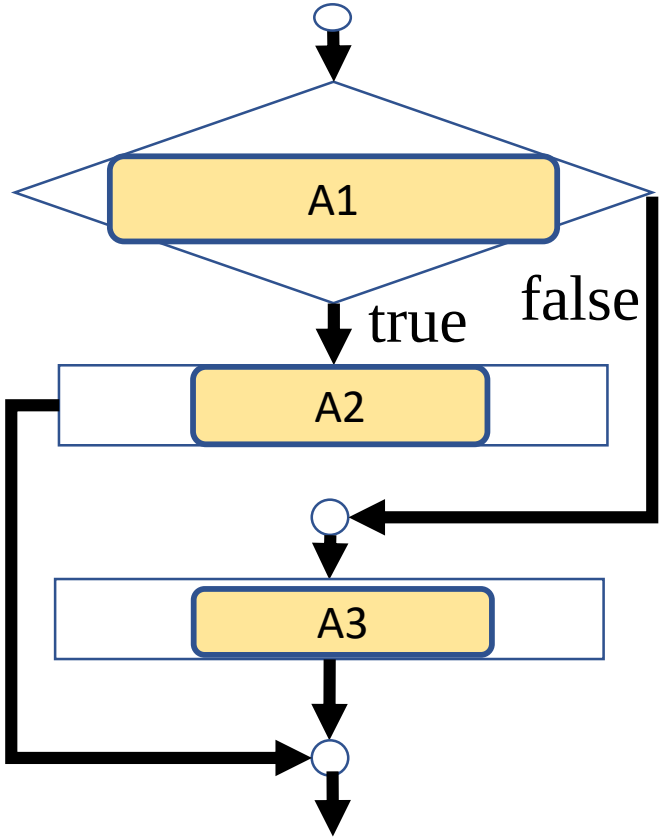
A non-zero value in the Boolean expression is interpreted as true.

Conditional Operator

```
(booleanExpression) ? expression1 : expression2

if (booleanExpression ) {

  expression1;

} else {

        expression2;

}
```

Conditional Operator

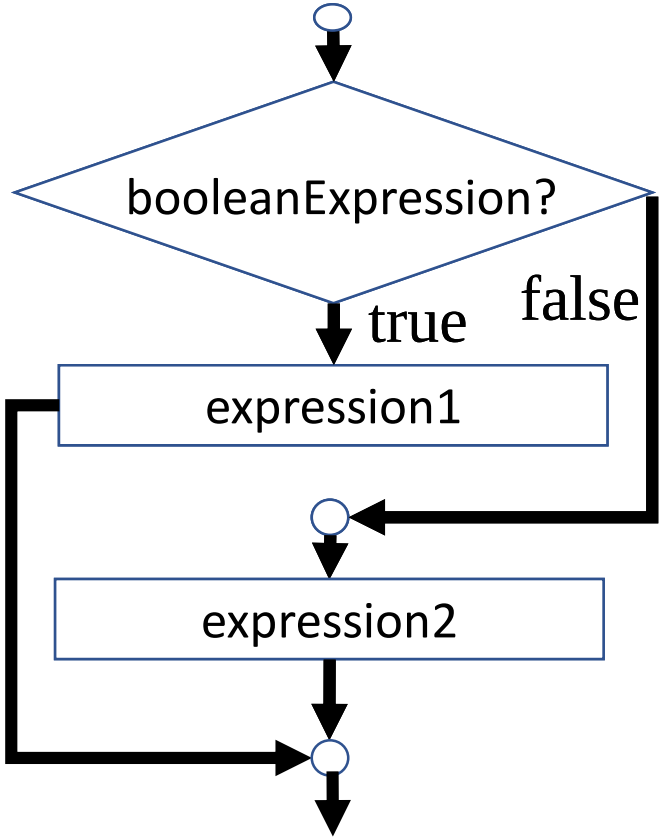
```
(booleanExpression) ? expression1 : expression2

if (booleanExpression ) {
  expression1;
} else {
        expression2;
}
```



Conditional Operator

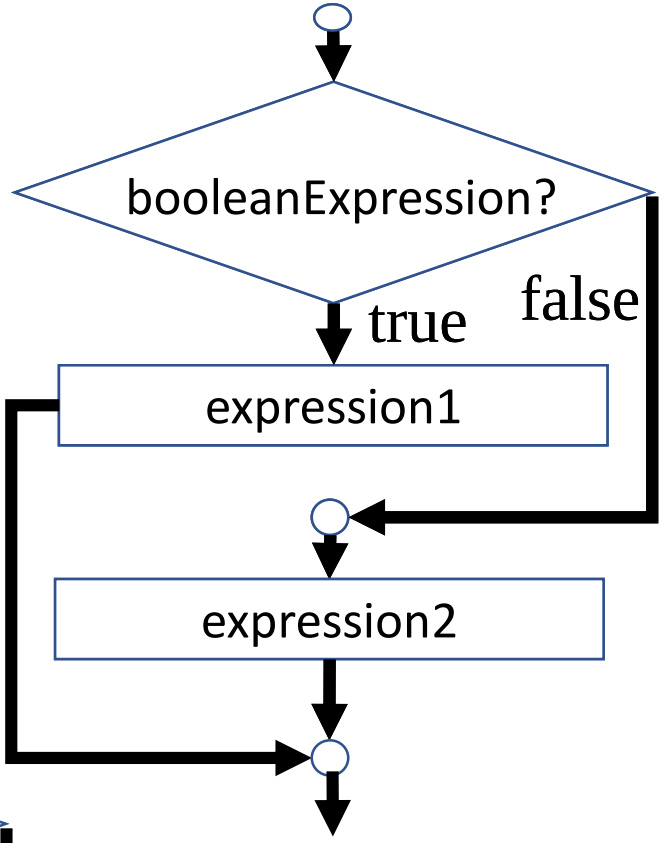
```
(booleanExpression) ? expression1 : expression2

if (booleanExpression ) {
  expression1;
} else {
        expression2;
}
```

```
x = (booleanExpression) ? expression1 : expression2;

If ( booleanExpression ) {
   x = expression1;
} else {
   x = expression2;
}
```

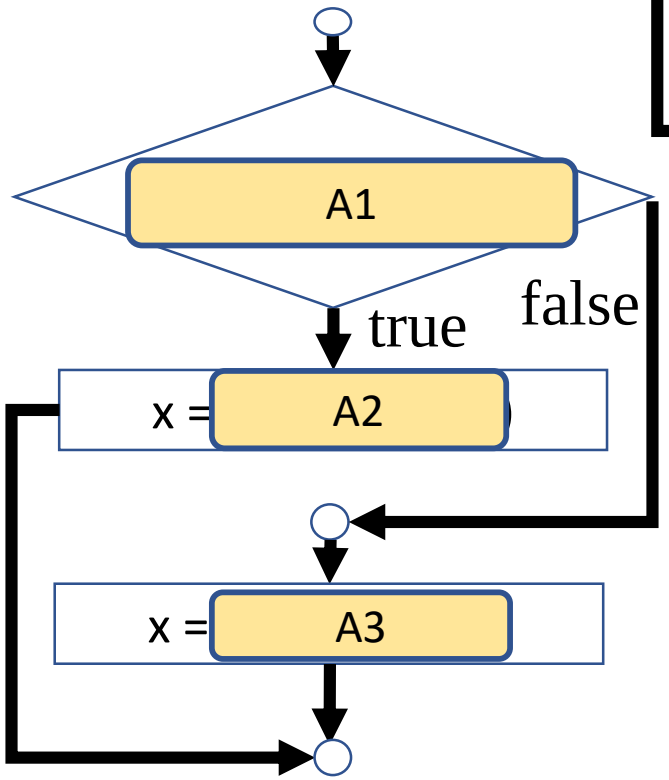


Conditional Operator

```
(booleanExpression) ? expression1 : expression2

if (booleanExpression ) {
  expression1;
} else {
      expression2;
}
```

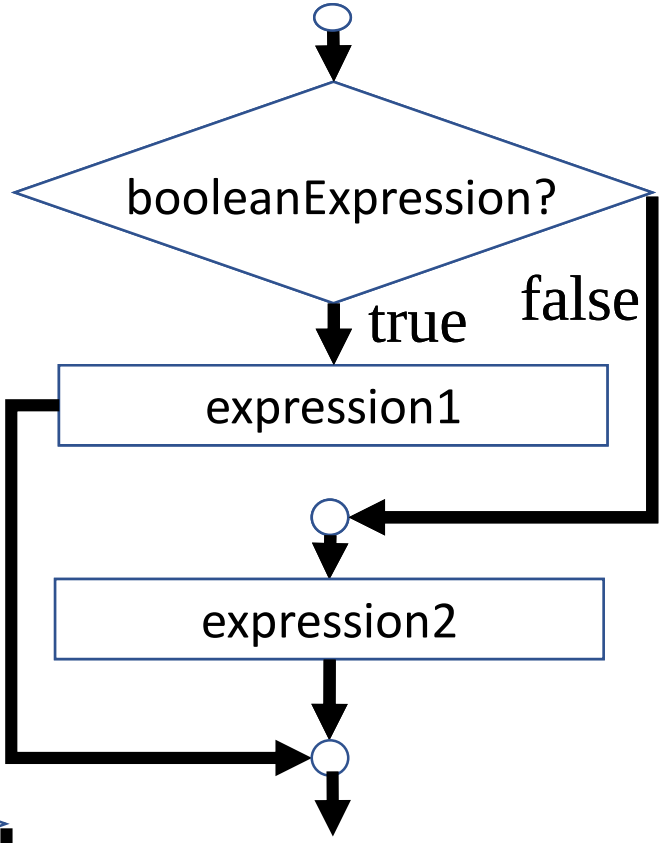



```
x = (booleanExpression) ? expression1 : expression2;

If ( booleanExpression ) {
   x = expression1;
} else {
   x = expression2;
}
```

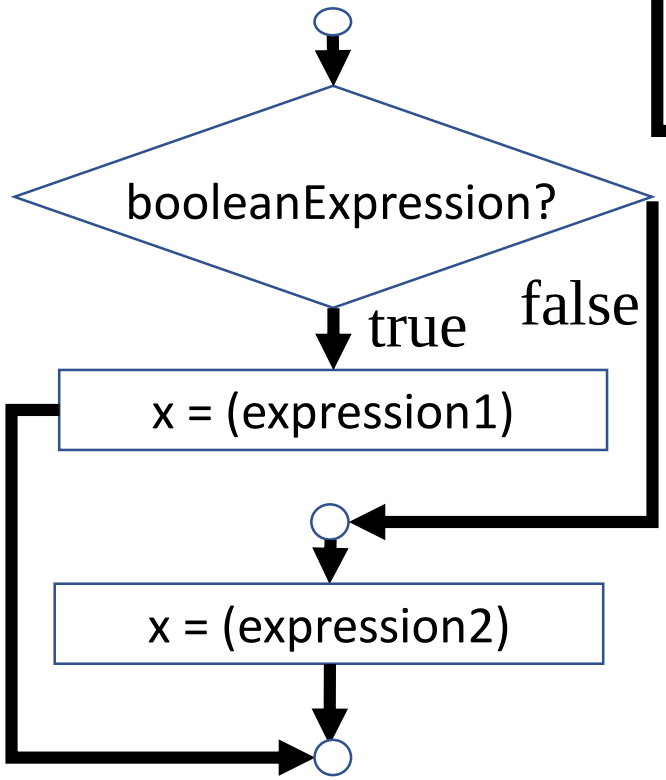


Conditional Operator

```
(booleanExpression) ? expression1 : expression2
if (booleanExpression ) {
  expression1;
} else {
        expression2;
}
```




```
x = (booleanExpression) ? expression1 : expression2;
If ( booleanExpression ) {
   x = expression1;
} else {
   x = expression2;
}
```



Conditional Operator

```
(booleanExpression) ? expression1 : expression2

if (booleanExpression ) {
  expression1;
} else {
      expression2;
}
```

```
x = (booleanExpression) ? expression1 : expression2;

If ( booleanExpression ) {
   x = expression1;
} else {
   x = expression2;
}
```

```
if (x > 0)
  y = 1;
else
  y = -1;
```

is equivalent to

y = (A) ? (B) : (C) ;

```
y = (x > 0) ?      1 :    -1;
```

What are (A), (B), and (C)?

Conditional Operator

```
(booleanExpression) ? expression1 : expression2

if (booleanExpression ) {
  expression1;
} else {
      expression2;
}
```

```
x = (booleanExpression) ? expression1 : expression2;

If ( booleanExpression ) {
   x = expression1;
} else {
   x = expression2;
}
```

```
if (x > 0)
  y = 1;
else
  y = -1;
```

is equivalent to

y = (A) ? (B) : (C)

```
y = (x > 0) ?      1 :    -1;
```

What are (A), (B), and (C)?

Ternary operator	? :	Need A1 operands
Binary operator	a + b	Need A2 operands
Unary operator	--x;	Need A3 operand

Conditional Operator

```
cout <<
  ((num % 2 == 0)
  ? "num is even"
  : "num is odd");
```

Interpret it as an if-else structure

Conditional Operator

```
cout <<
 ((num % 2 == 0)
 ? "num is even"
 : "num is odd");
```

Interpret it as an if-else structure

```
If ( num % 2 == 0 ) {
  cout <<    A1    ;
} else {
  cout <<  A2
}
```

while, do-while, and for-loop

Repetition structures

while and do-while

```
while (expression ) {
   //body is executed if expression is true
}



do {
   //body must be executed one time
} while (expression);
```

```
while ( a> 0) {
        …
}



do {
        …
} while (a > 0);
```

while and do-while

```
while (expression ) {
   //body is executed if expression is true
}



do {
   //body must be executed one time
} while (expression);
```

```
int s = 0, a = 0;
while ( a > 0) {
        s = s + 1; a = a -1;
}
// what is the value of s?

int s = 0, a = 0;
do {
       s = s + 1; a = a -1;
} while (a > 0);
// what is the value of s?
```

while and do-while

```
while (expression ) {
   //body is executed if expression is true
}



do {
   //body must be executed one time
} while (expression);
```

```
int s = 0, a = 0;
while ( a > 0) {
        s = s + 1; a = a -1;
}
// what is the value of s?

unsigned int s = 0, a = 0;
do {
       s = s + 1; a = a -1;
} while (a > 0);
// what is the value of s?
```

while and do-while

```
while (expression ) {
   //body is executed if expression is true
}



do {
   //body must be executed one time
} while (expression);
```

```
int s = 0, a = 0;
while ( a > 0) {
      s = s + 1; a = a -1;
}
// what is the value of s?

unsigned int s = 0, a = 0;
do {
      s = s + 1; a = a -1;
} while (a > 0);
// what is the value of s?
```

Implement the program and display s.

for-loop

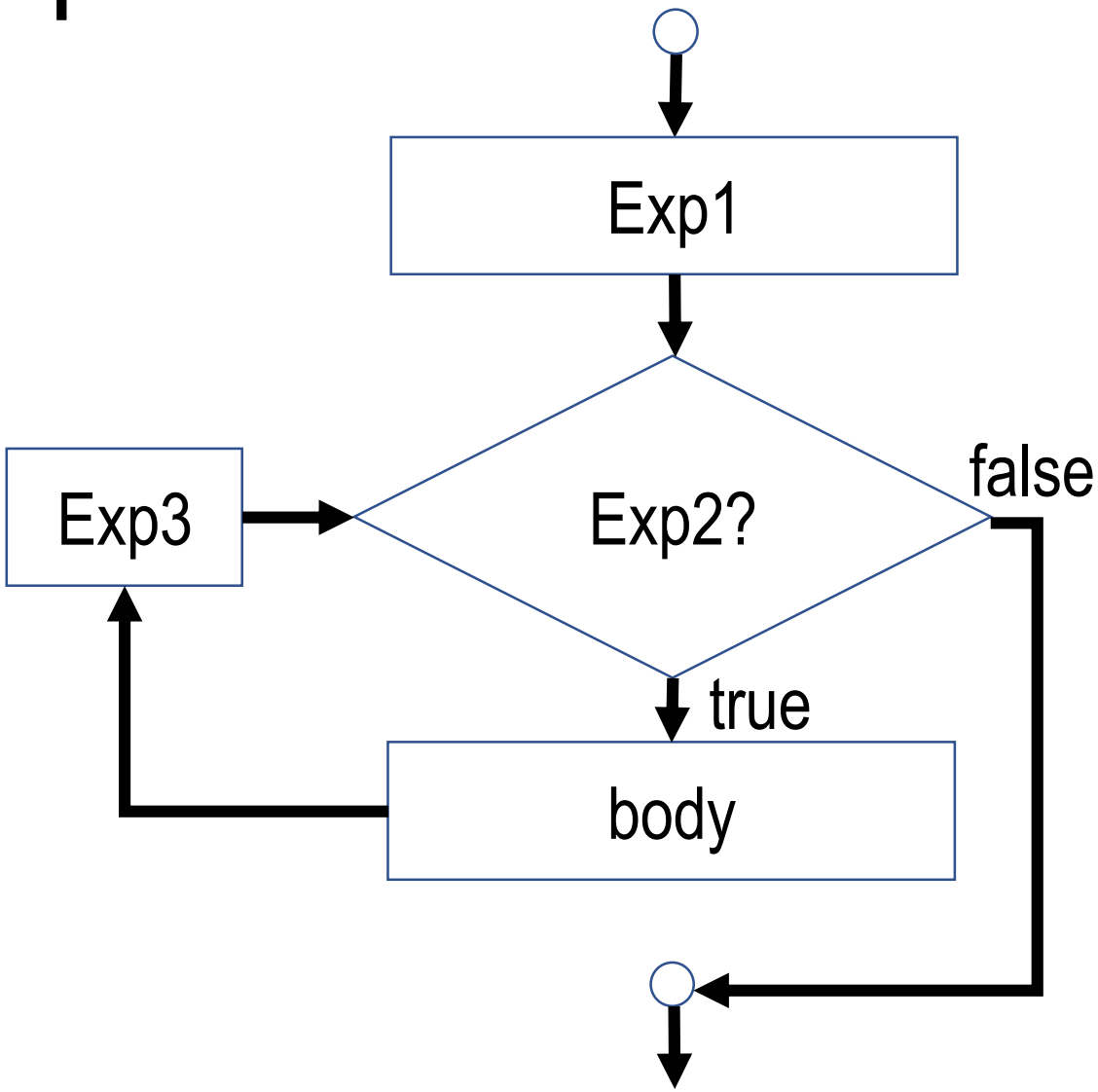
```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2



for-loop

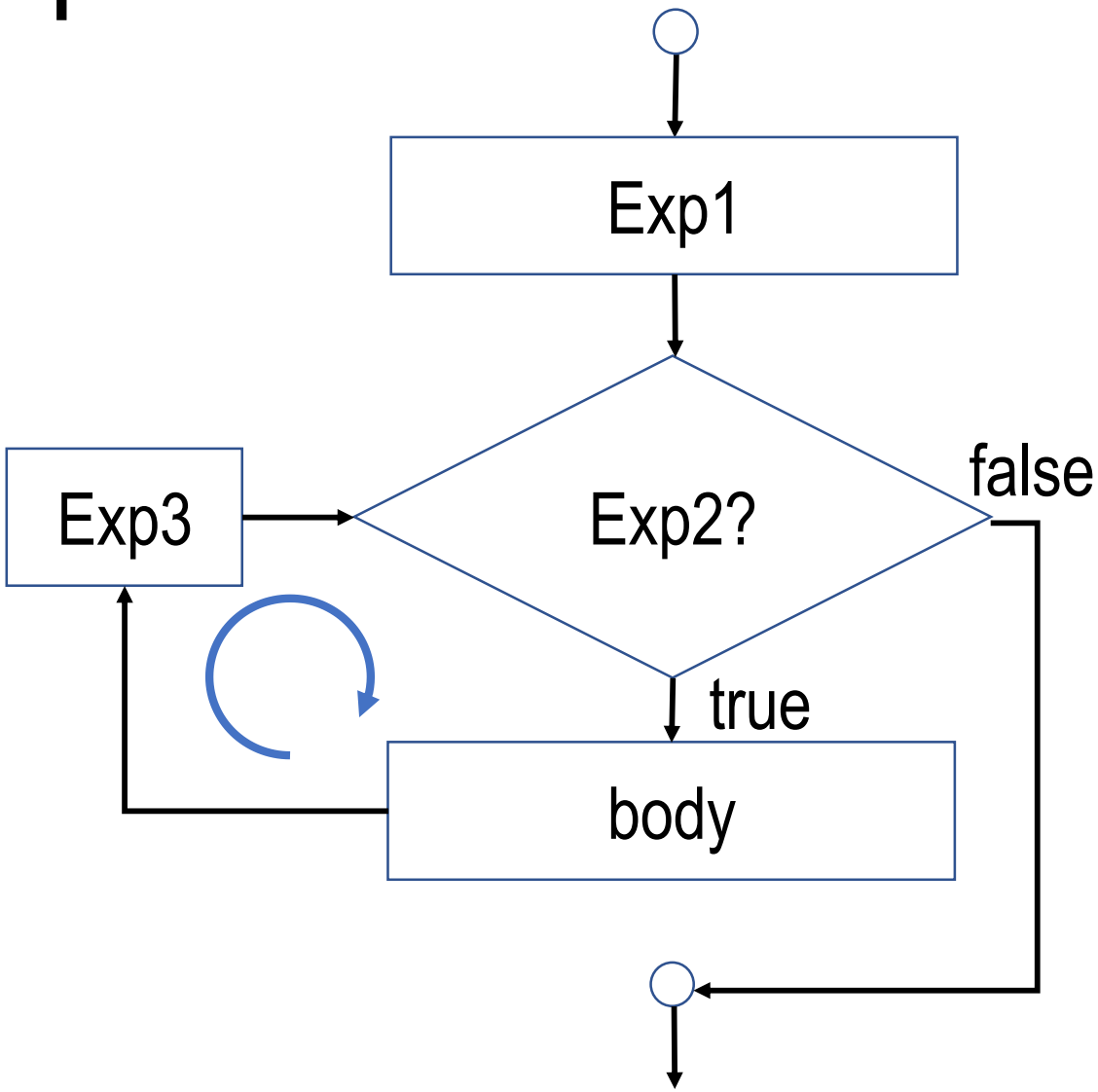
```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2



for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0;
    i < 10;
    ++i ) {
    s += 2;
}
```

What is the value of s after the fragment is executed?

for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

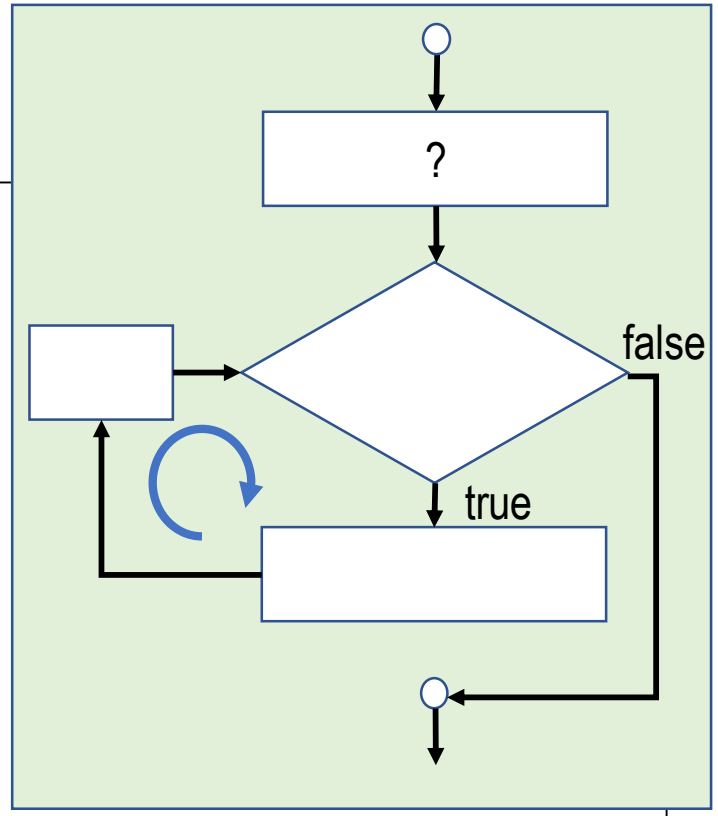
Exp1: initialization expression

Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0;
     i < 10;
     ++i ) {
     s += 2;
}
```



for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

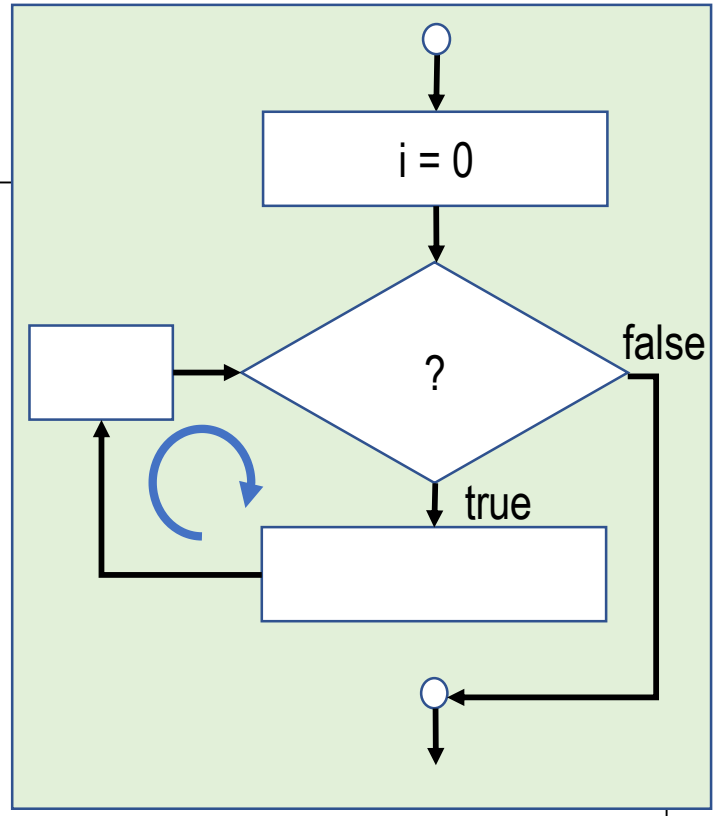
Exp1: initialization expression

Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0;
     i < 10;
     ++i ) {
     s += 2;
}
```



for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

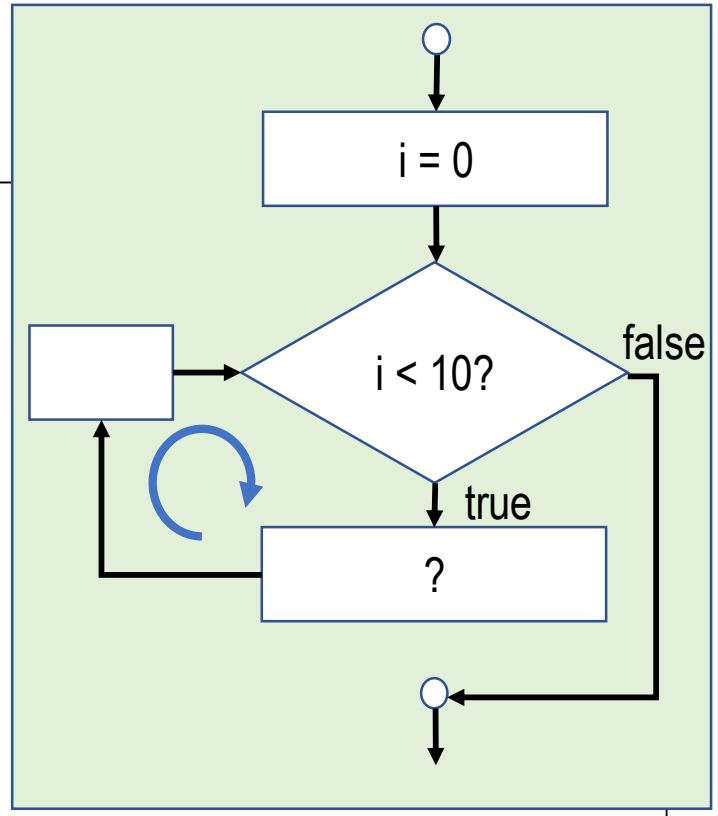
Exp1: initialization expression

Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0;
    i < 10;
    ++i ) {
    s += 2;
}
```



for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

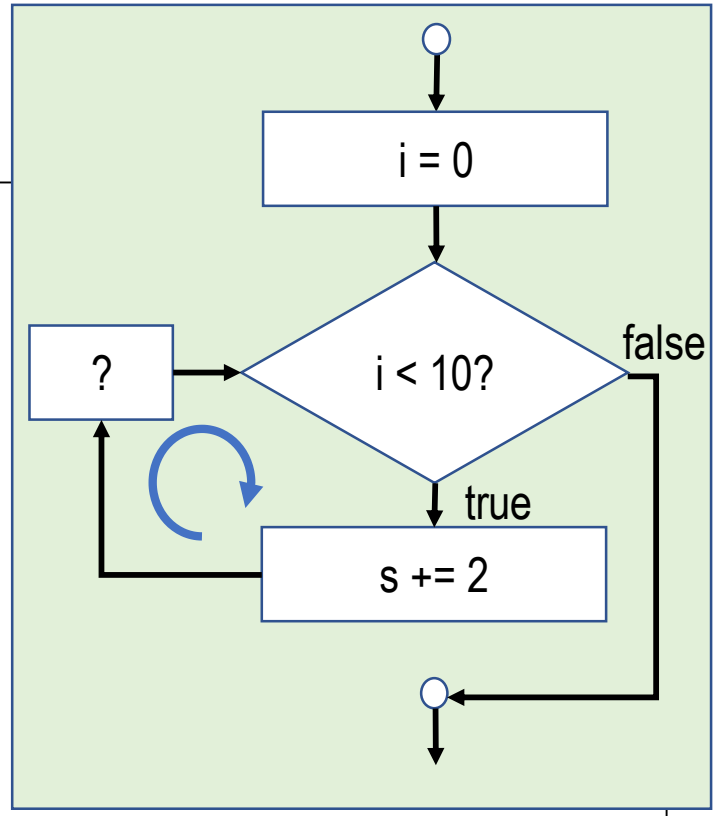
Exp1: initialization expression

Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0;
    i < 10;
    ++i ) {
    s += 2;
}
```



for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

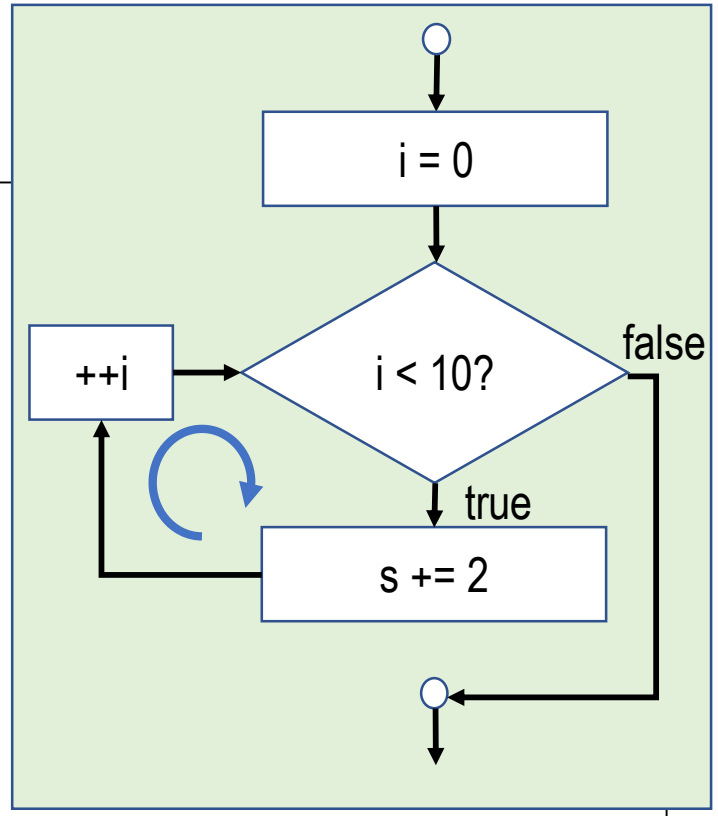
Exp1: initialization expression

Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0;
     i < 10;
     ++i ) {
     s += 2;
}
```



What is the value of s after the fragment is executed?

for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

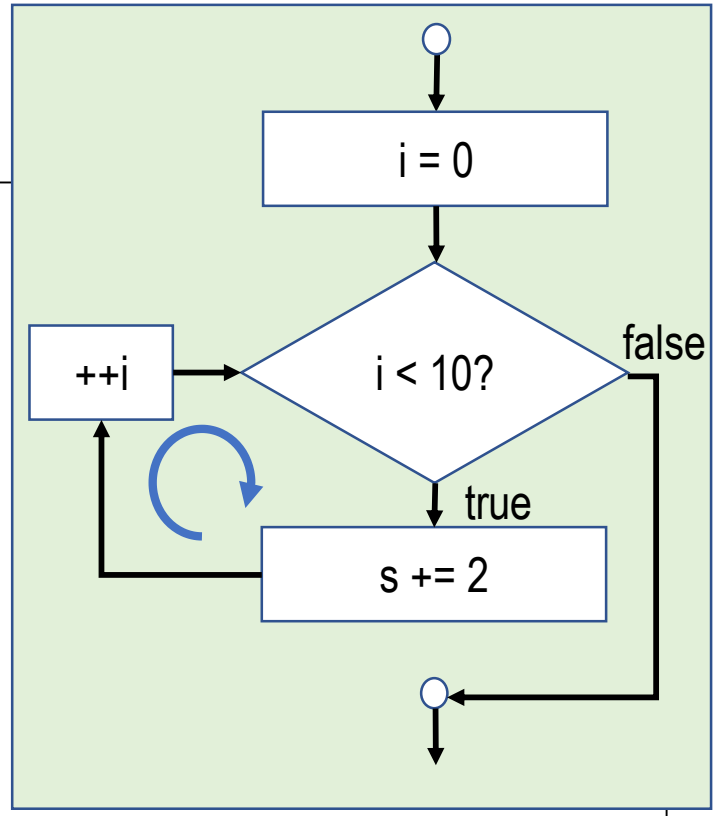
Exp1: initialization expression

Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0;
     i < 10;
     ++i ) {
     s += 2;
}
```



What is the value of s after the fragment is executed?

The body is executed for A1 times.
i =0, 1, 2, …, 9. s = A2

for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0, j = 1;
    ;
    ++i ) {
    if ( i > 20) break;
    s += 2 + j;
    j = j + 2;
}
```

What is the value of s after the fragment is executed?

for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

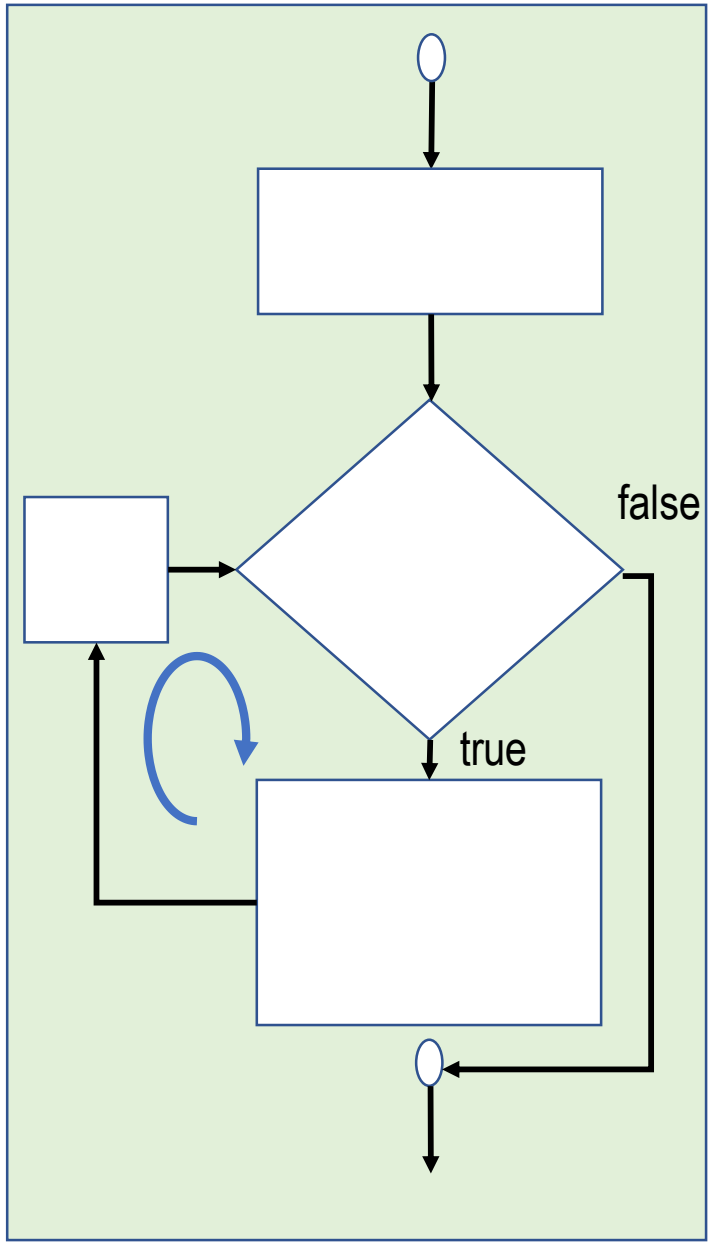
Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0, j = 1;
      ;
      ++i ) {
      if ( i > 20) break;
      s += 2 + j;
      j = j + 2;
}
```

What is the value of s after the fragment is executed?



for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

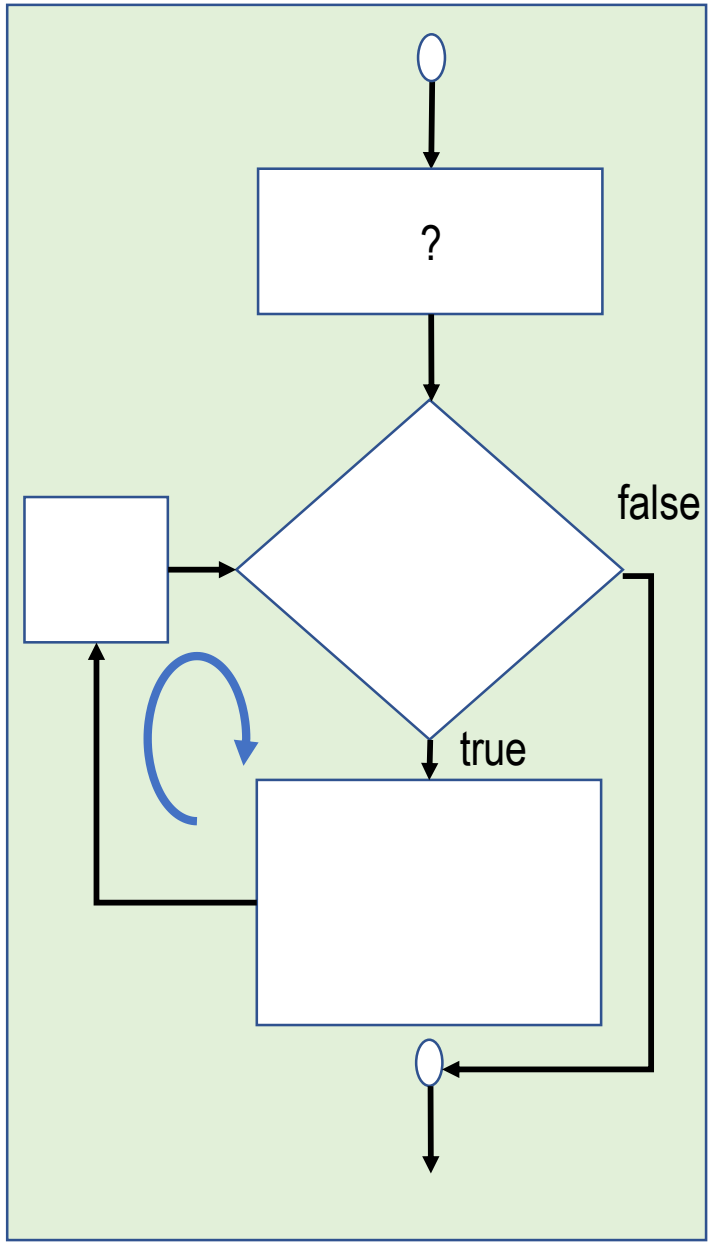
Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0, j = 1;
      ;
      ++i ) {
      if ( i > 20) break;
      s += 2 + j;
      j = j + 2;
}
```

What is the value of s after the fragment is executed?



for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

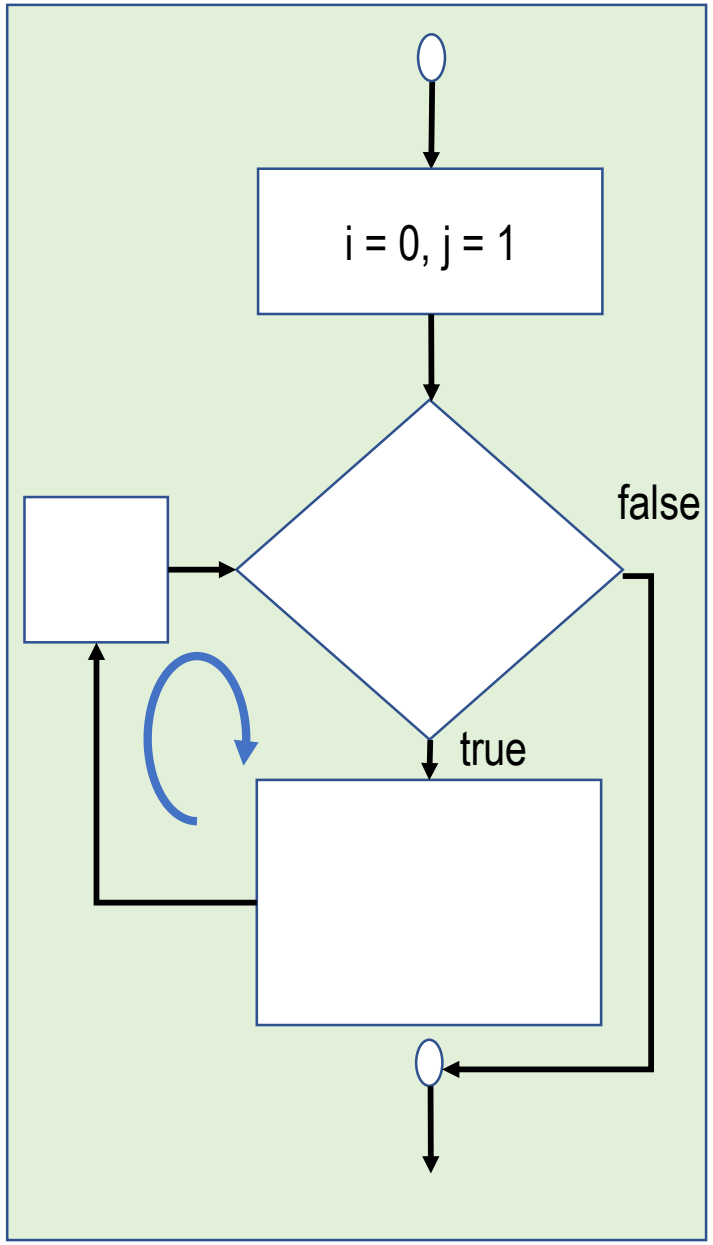
Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0, j = 1;
    ;
    ++i ) {
    if ( i > 20) break;
    s += 2 + j;
    j = j + 2;
}
```

What is the value of s after the fragment is executed?



for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

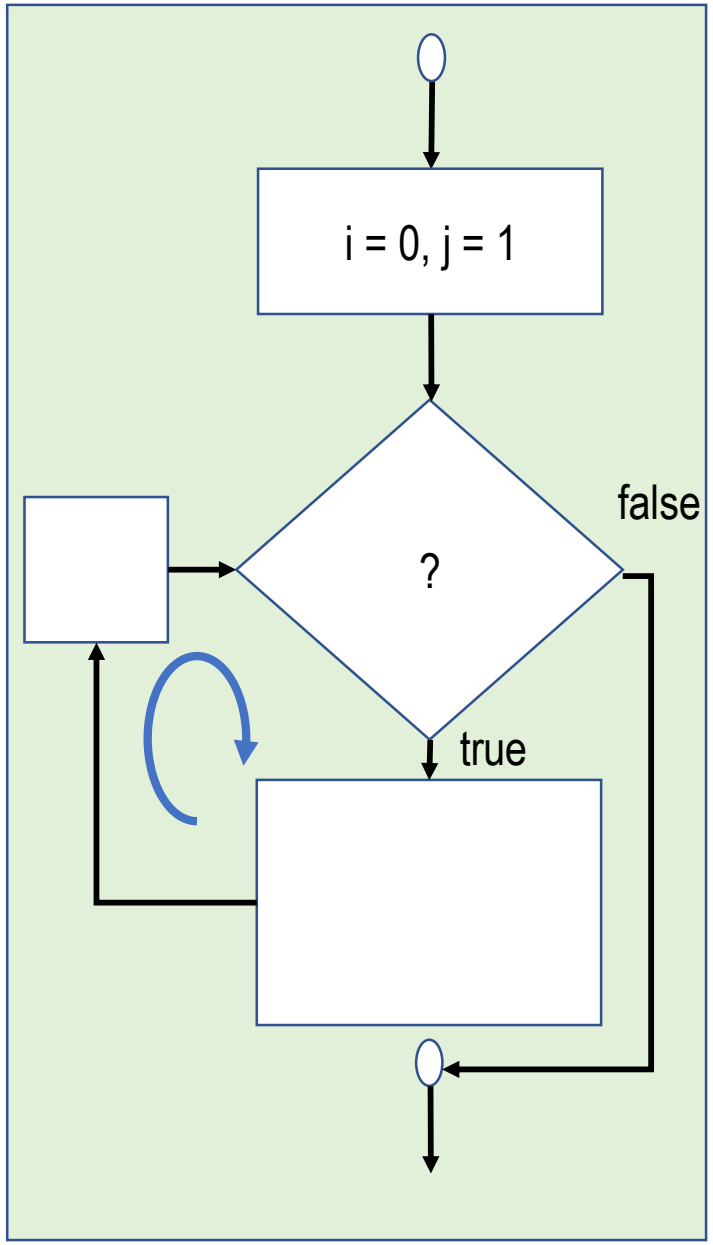
Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0, j = 1;
      ;
      ++i ) {
      if ( i > 20) break;
      s += 2 + j;
      j = j + 2;
}
```

What is the value of s after the fragment is executed?



for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

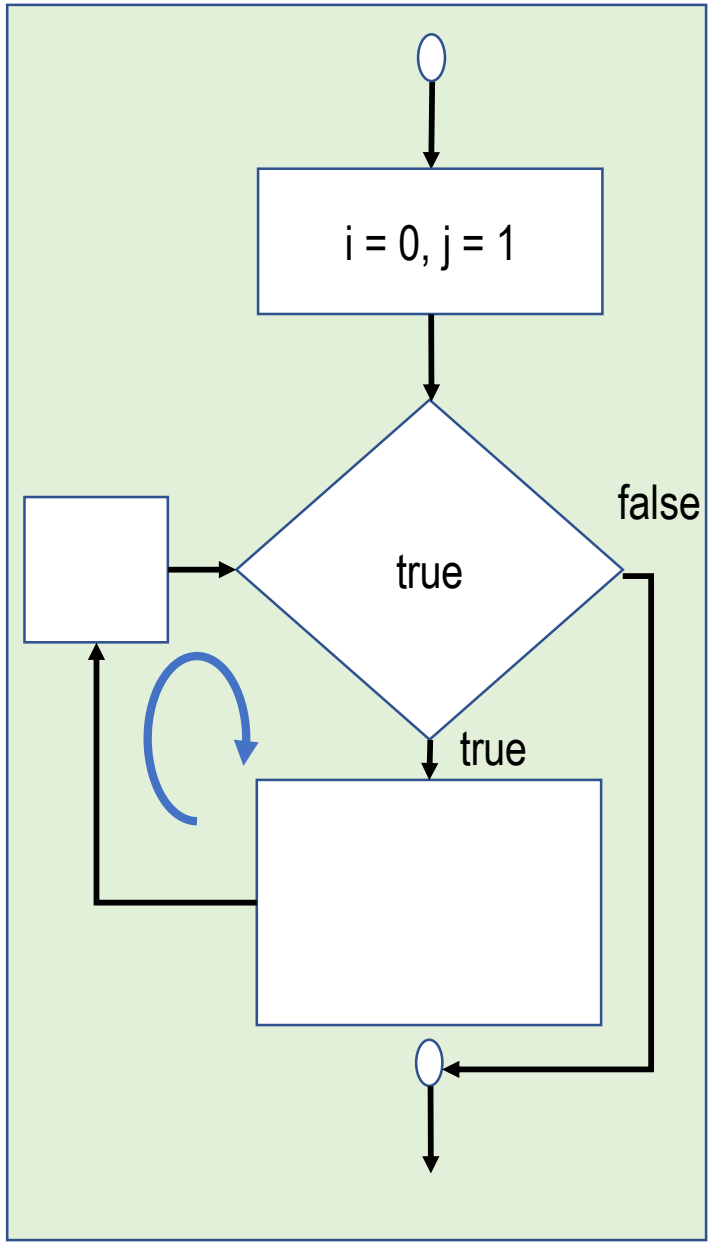
Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0, j = 1;
    ;
    ++i ) {
    if ( i > 20) break;
    s += 2 + j;
    j = j + 2;
}
```

What is the value of s after the fragment is executed?



for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

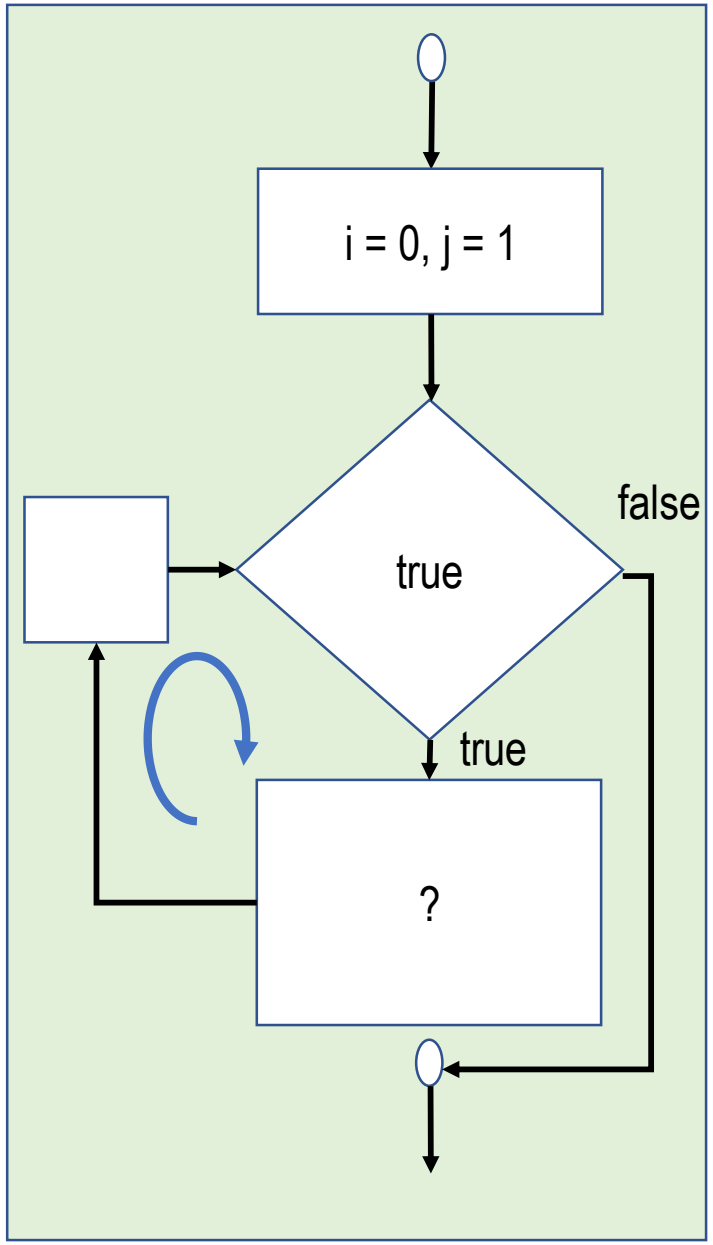
Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0, j = 1;
    ;
    ++i ) {
    if ( i > 20) break;
    s += 2 + j;
    j = j + 2;
}
```

What is the value of s after the fragment is executed?



for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

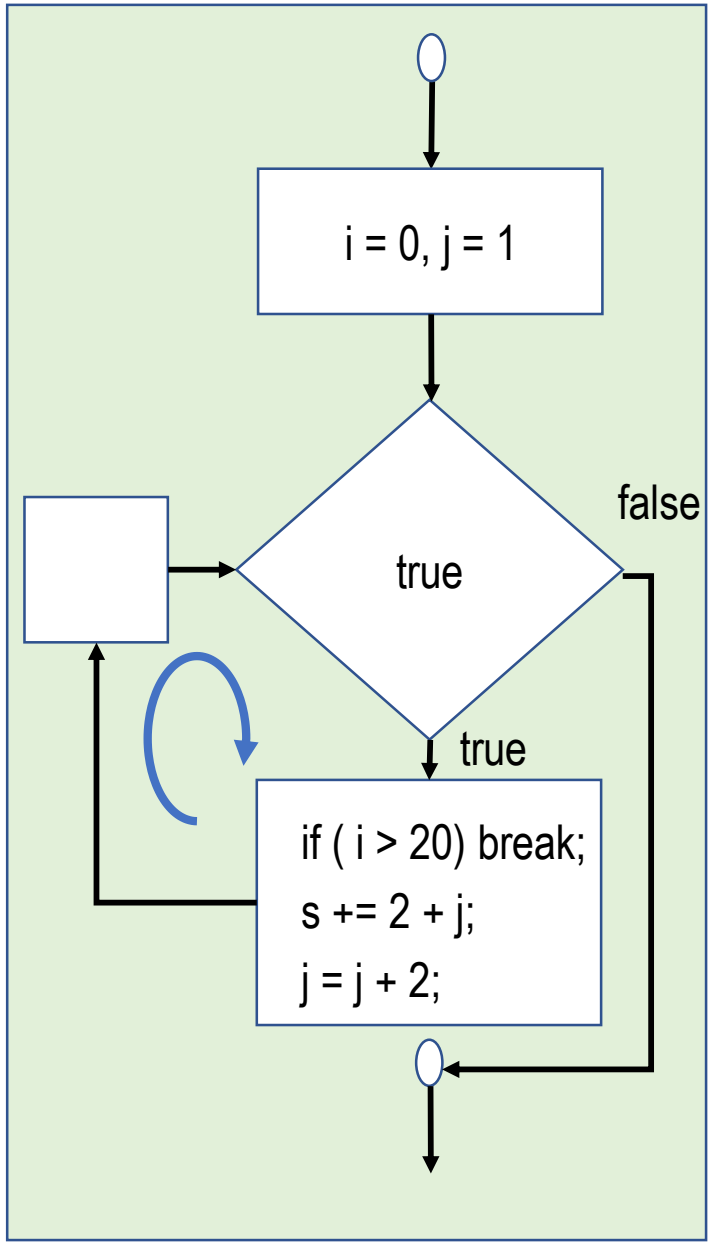
Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0, j = 1;
    ;
    ++i ) {
    if ( i > 20) break;
    s += 2 + j;
    j = j + 2;
}
```

What is the value of s after the fragment is executed?



for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

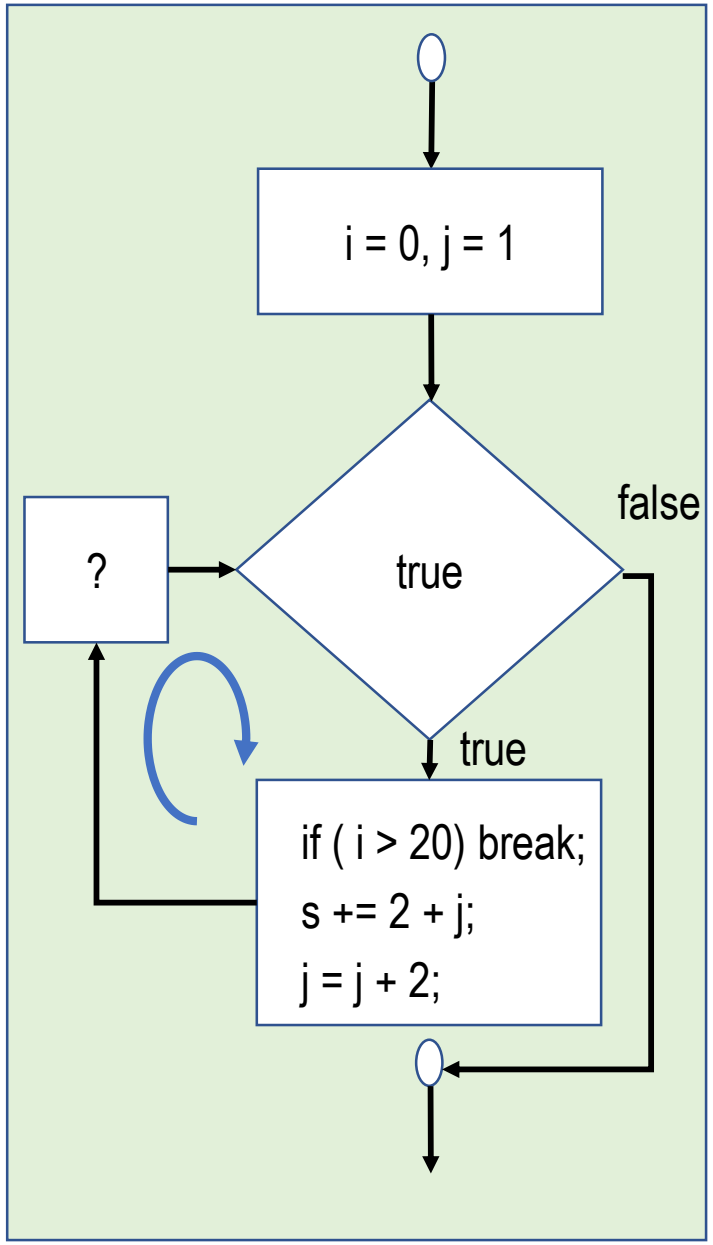
Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0, j = 1;
    ;
    ++i ) {
    if ( i > 20) break;
    s += 2 + j;
    j = j + 2;
}
```

What is the value of s after the fragment is executed?



for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

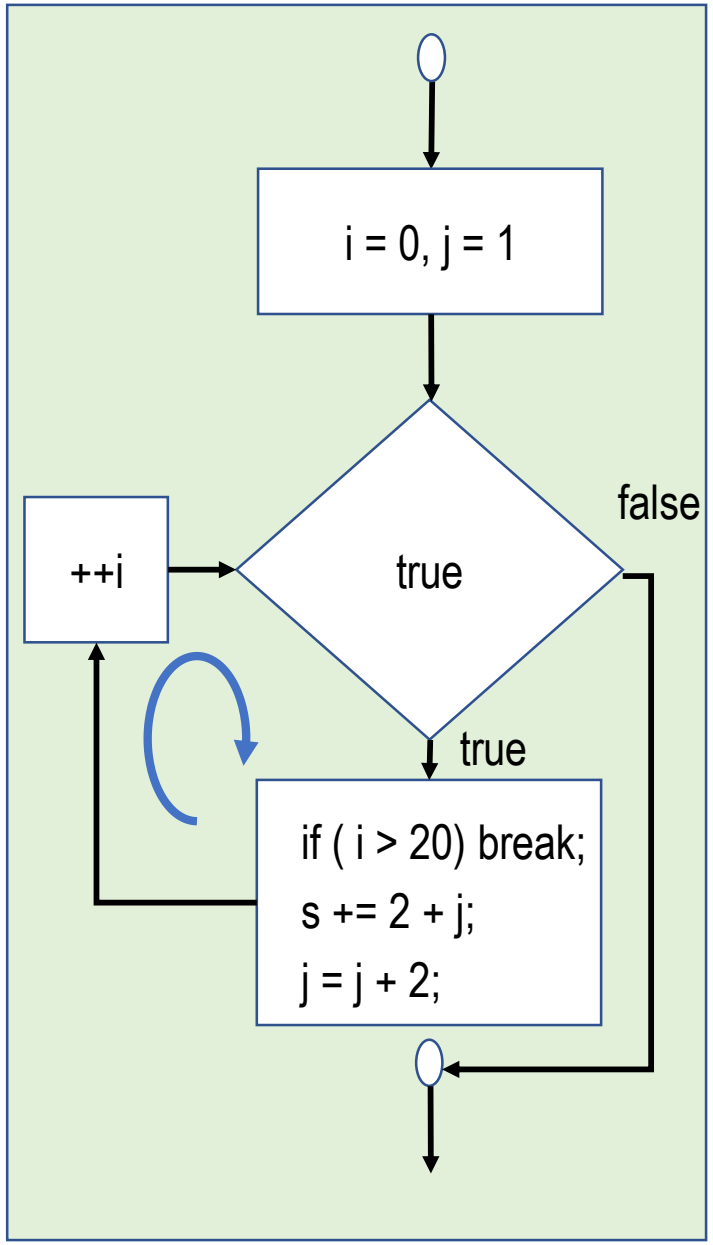
Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0, j = 1;
    ;
    ++i ) {
    if ( i > 20) break;
    s += 2 + j;
    j = j + 2;
}
```

What is the value of s after the fragment is executed?



Implement a program and check your answer.

for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

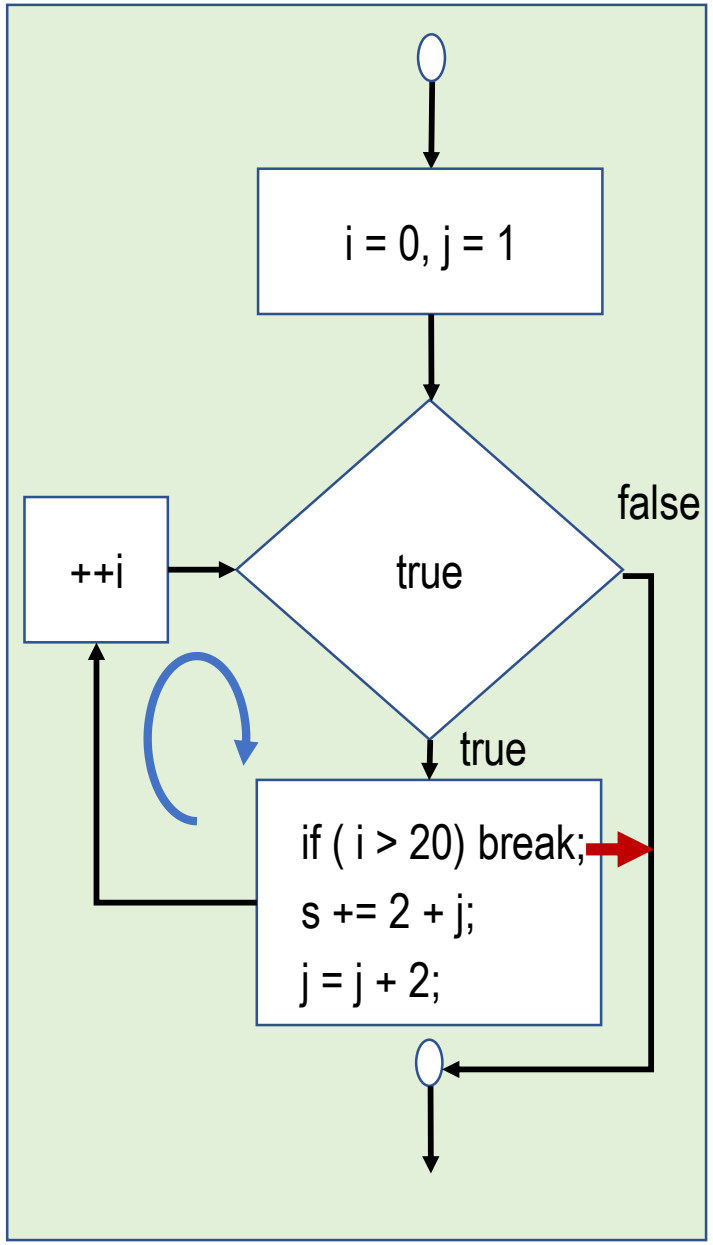
Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0, j = 1;
    ;
    ++i ) {
    if ( i > 20) break;
    s += 2 + j;
    j = j + 2;
}
```

What is the value of s after the fragment is executed?



Implement a program and check your answer.

for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

Infinite loops

```
for (int i =0; ;) { }

int j = 0;
for (;;++j) { }

for (;;);
```

for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

Infinite loops

```
for (int i =0; ;) { }

int j = 0;
for (;;++j) { }

for (;;);
```

Expr2 is empty.
Its value is true.

switch blocks

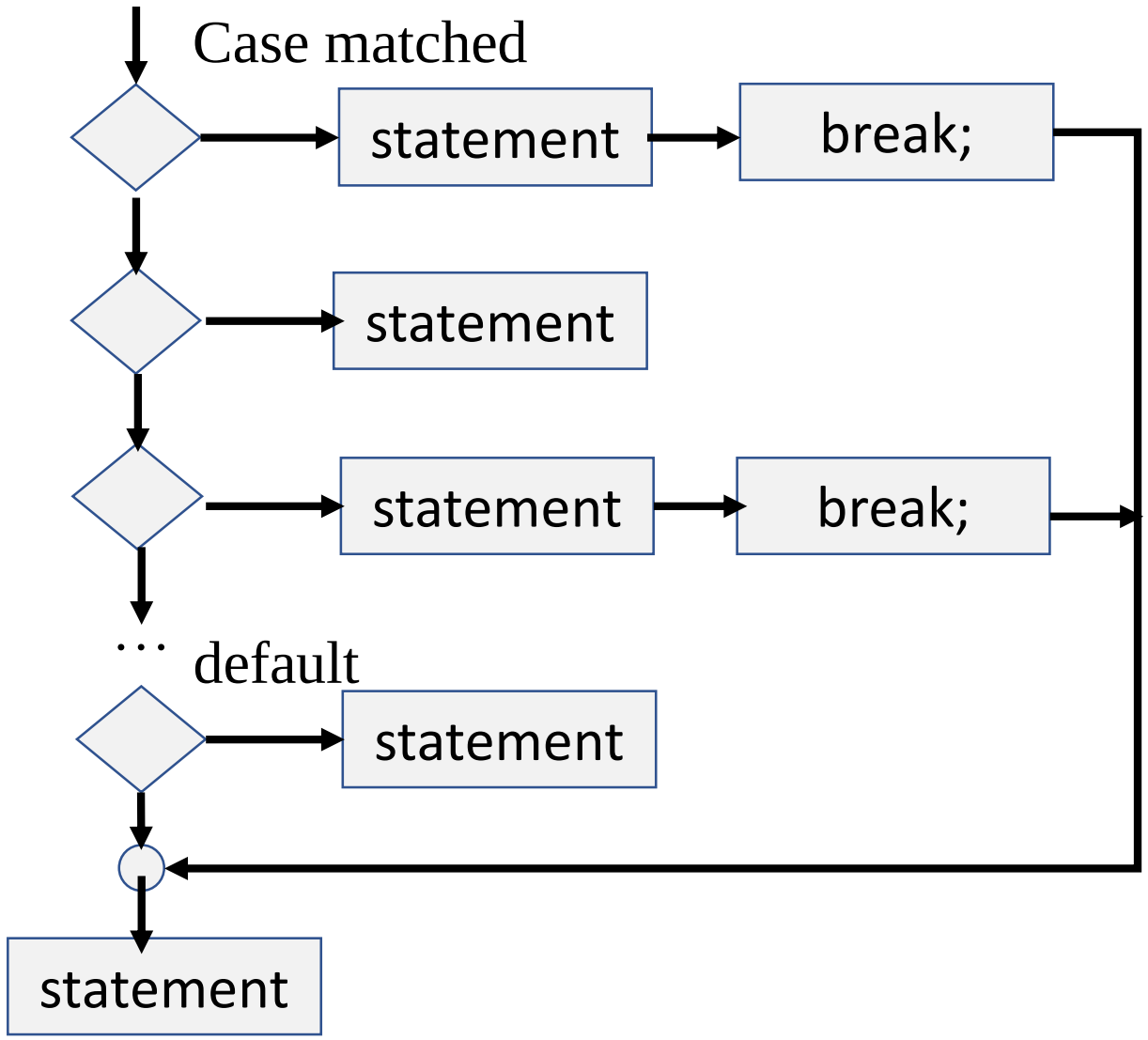
Fall through mechanism

The switch-structure

```
switch (grade)
{
  case 0:  cout << “A+” << endl;
           break;
  case 1:  cout << “A” << endl;
           break;
  case 2:  cout << “A-” << endl;
           break;
  case 3:  cout << “B+” << endl;
           break;
  default: cout << “Below B+“ << endl;
}
```

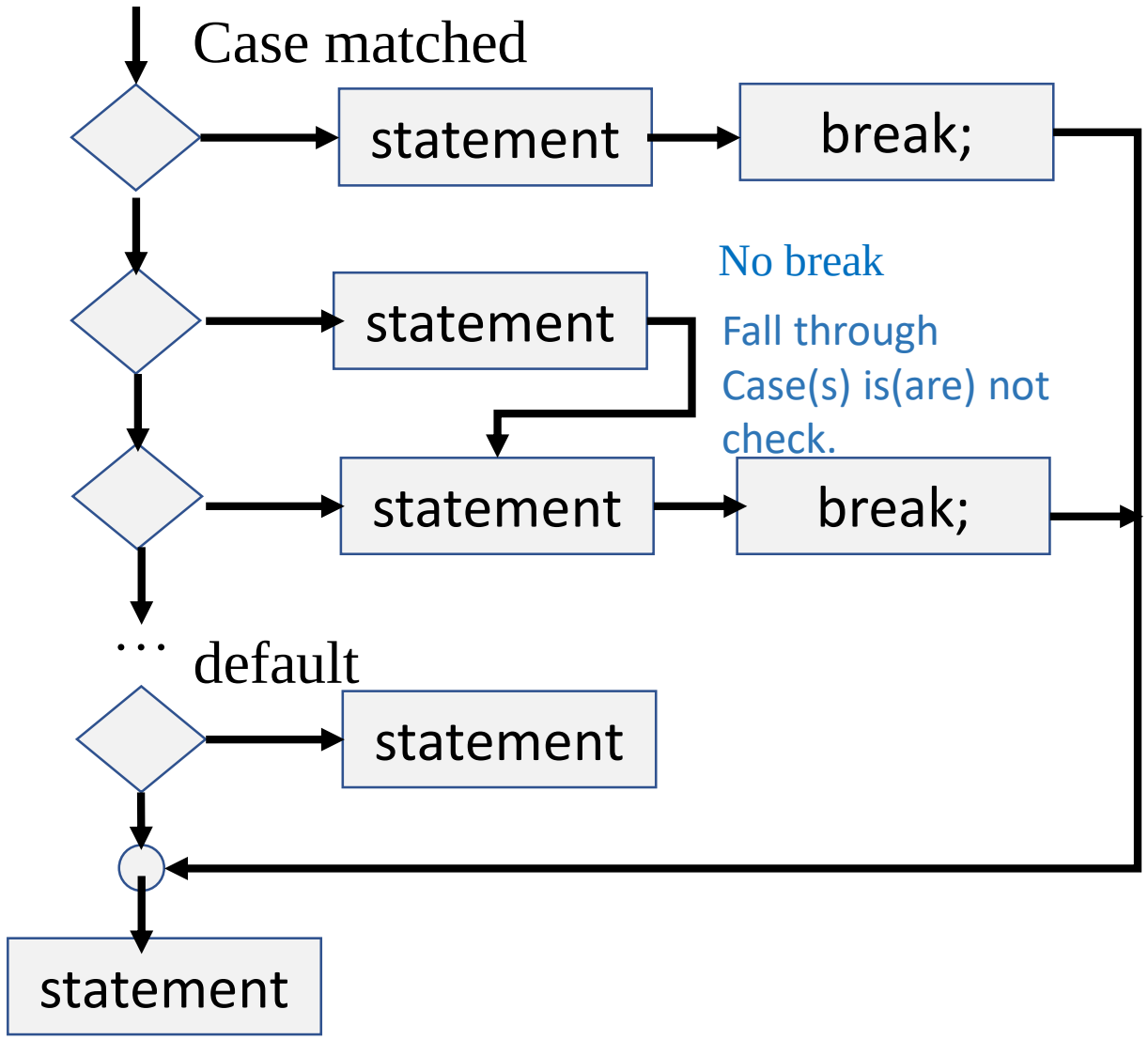
The switch-structure
Flow chart

```
switch (grade)
{
  case 0:  cout << "A+" << endl;
           break;
  case 1:  cout << "A" << endl;
           break;
  case 2:  cout << "A-" << endl;
           break;
  case 3:  cout << "B+" << endl;
           break;
  default: cout << "Below B+" << endl;
}
```



The switch-structure
Flow chart

```
switch (grade)
{
  case 0:  cout << "A+" << endl;
           break;
  case 1:  cout << "A" << endl;
           break;
  case 2:  cout << "A-" << endl;
           break;
  case 3:  cout << "B+" << endl;
           break;
  default: cout << "Below B+" << endl;
}
```



Dry run

grade = 2

grade is not equal to 0

```
switch (grade)
{
  case 0:  cout << “A+” << endl;
           break;
  case 1:  cout << “A” << endl;
           break;
  case 2:  cout << “A-” << endl;
           break;
  case 3:  cout << “B+” << endl;
           break;
  default: cout << “Below B+“ << endl;
}
```

Dry run

grade = 2

grade is not equal to 1

```
switch (grade)
{
  case 0:  cout << “A+” << endl;
           break;
  case 1:  cout << “A” << endl;
           break;
  case 2:  cout << “A-” << endl;
           break;
  case 3:  cout << “B+” << endl;
           break;
  default: cout << “Below B+“ << endl;
}
```

Dry run

grade = 2

grade is equal to 2

```
switch (grade)
{
  case 0:  cout << “A+” << endl;
           break;
  case 1:  cout << “A” << endl;
           break;
  case 2:  cout << “A-” << endl;
           break;
  case 3:  cout << “B+” << endl;
           break;
  default: cout << “Below B+“ << endl;
}
```

Dry run

grade = 2

```
switch (grade)
{
  case 0:  cout << “A+” << endl;
           break;
  case 1:  cout << “A” << endl;
           break;
  case 2:  cout << “A-” << endl;
           break;
  case 3:  cout << “B+” << endl;
           break;
  default: cout << “Below B+“ << endl;
}
```

grade is equal to 2

Output
A-

Dry run

grade = 2

```
switch (grade)
{
  case 0:  cout << “A+” << endl;
           break;
  case 1:  cout << “A” << endl;
           break;
  case 2:  cout << “A-” << endl;
           break;
  case 3:  cout << “B+” << endl;
           break;
  default: cout << “Below B+“ << endl;
}
```

Exit the switch-structure

Dry run

grade = 2

```
switch (grade)
{
  case 0:  cout << “A+” << endl;
           break;
  case 1:  cout << “A” << endl;
           break;
  case 2:  cout << “A-” << endl;
           break;
  case 3:  cout << “B+” << endl;
           break;
  default: cout << “Below B+“ << endl;
}
```

Exit the switch block.

Dry run

grade = 0

```
switch (grade)
{
  case 0:  // fall through
           cout << “A+” << endl;
  case 1:  // fall through
           cout << “A or higher” << endl;
  case 2:  // fall through
  case 3:  // fall through
          cout << “B+ or higher” << endl;
          break;
  default: cout << “Below B+“ << endl;
}
```

What are the output?

Dry run

grade = 0

```
switch (grade)
{
  case 0:  // fall through
           cout << “A+” << endl;
  case 1:  // fall through
           cout << “A or higher” << endl;
  case 2:  // fall through
  case 3:  // fall through
          cout << “B+ or higher” << endl;
          break;
  default: cout << “Below B+“ << endl;
}
```

What are the output?

A+
A or higher
B+ or higher

Fall through: Once a case is satisfied, ignore to check the remaining case values.

Logical operators

Truth tables and short-circuits

Logical operators

Operator	Name	Purpose
!	not	negation
&&	and	logical conjunction
‖	or	logical disjunction

Logical operators connect multiple conditions.

Truth Table: negation

!x

x		!x
false		A1
true		A2

The result is the opposite of the input.

Truth Table: logical conjunction

x && y

x	y	(x && y)
false	false	
false	true	A1
true	false	
true	true	A2

If both of them are true, the result is true. Otherwise, the result is false.

Truth Table: logical disjunction

x || y

x	y	(x || y)
false	false	A1
false	true	A2
true	false	
true	true	

If one of them is true, the result is true. Otherwise, the result is false.

Short-Circuit Operator

```
if ( x && y) {
   statements;
}
```

For evaluating x && y:

First, evaluates x

If x is true and then evaluates y

If y is true, then execute statements

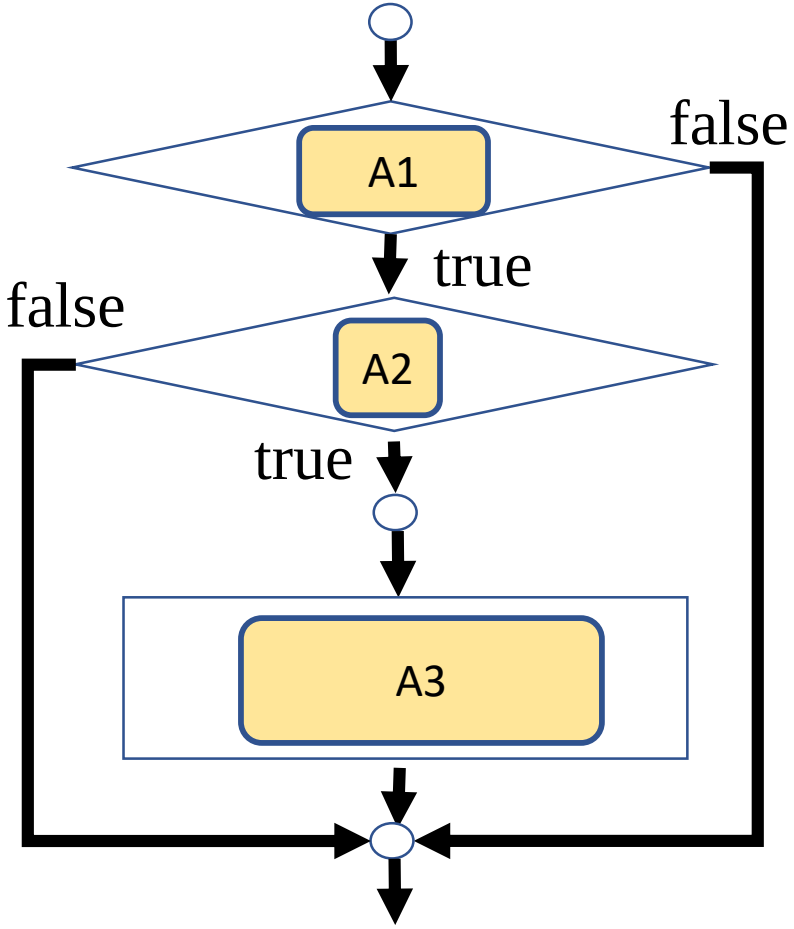
If x is false, y is not evaluated.

Short-Circuit Operator

```
if ( x && y) {
   statements;
}
```

For evaluating x && y:
First, evaluates x
    If x is true and then evaluates y
        If y is true, then execute statements

    If x is false, y is not evaluated.

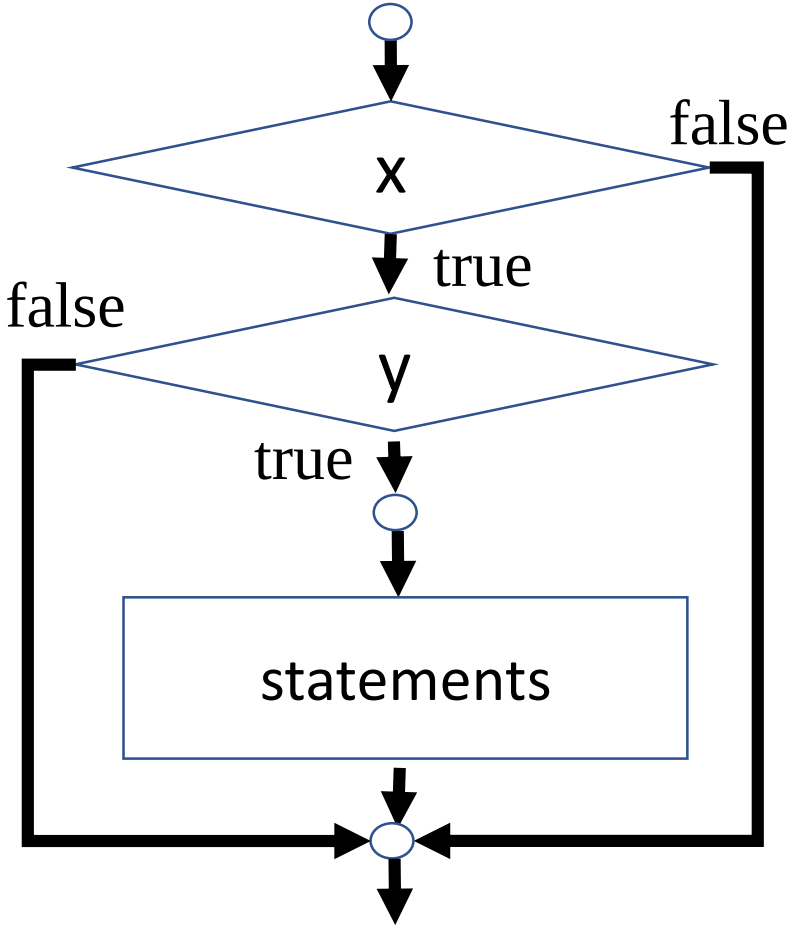


Short-Circuit Operator

```
if ( x && y) {
   statements;
}
```

For evaluating x && y:
First, evaluates x
    If x is true and then evaluates y
        If y is true, then execute statements

    If x is false, y is not evaluated.

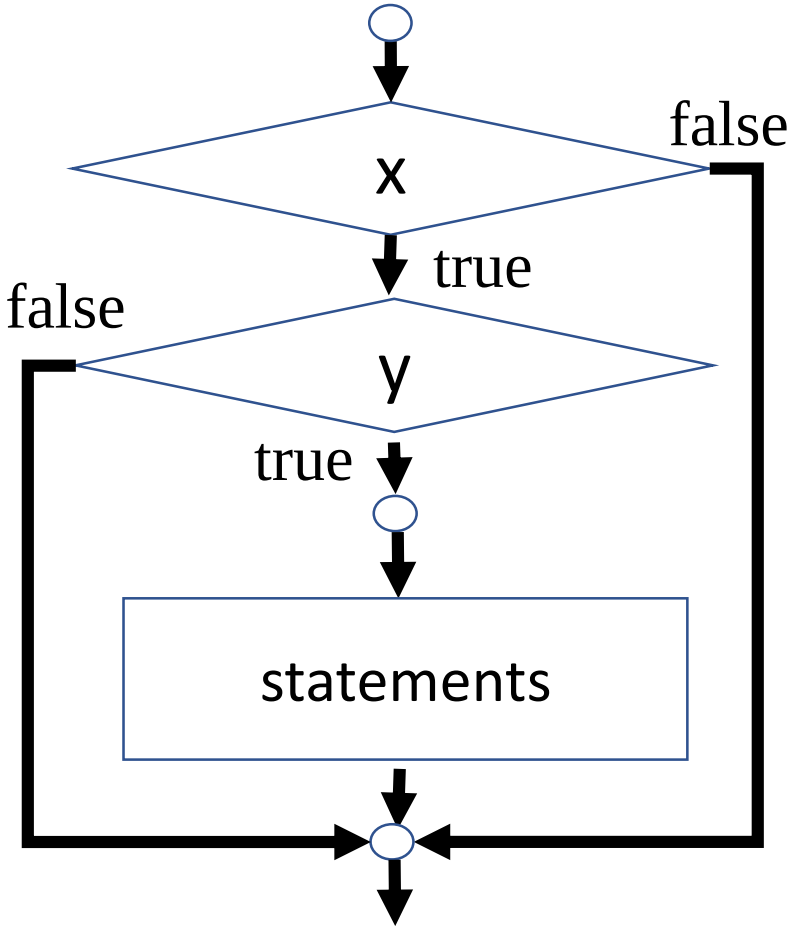


Short-Circuit Operator

```
if ( x && y) {
   statements;
}
```

For evaluating x && y:
First, evaluates x
    If x is true and then evaluates y
        If y is true, then execute statements

    If x is false, y is not evaluated.



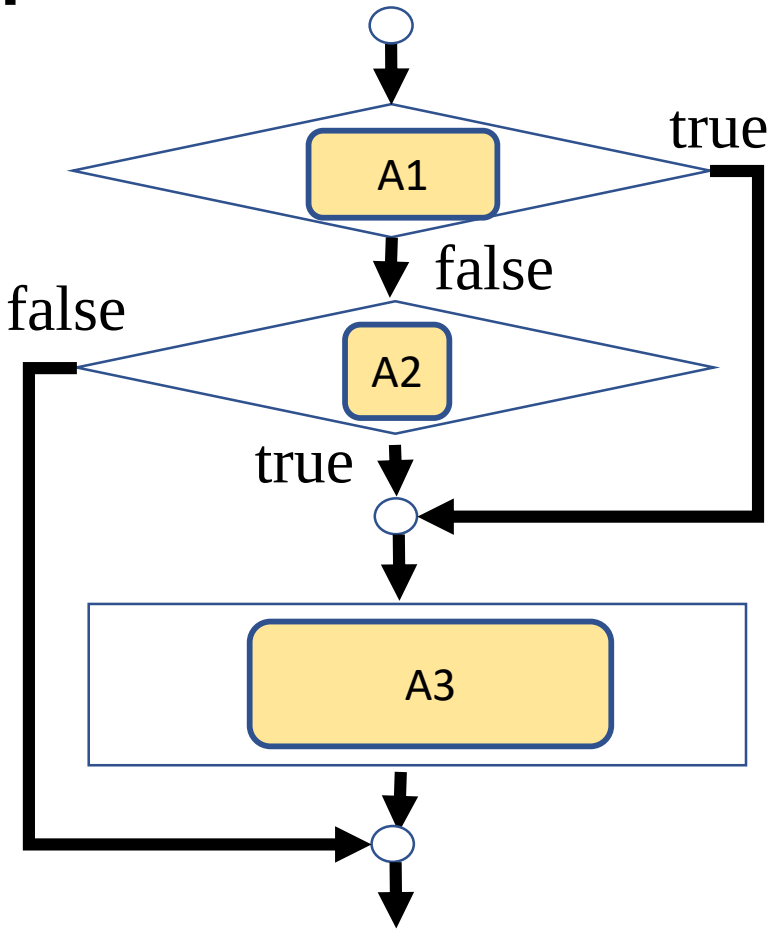
Note:	x and y can be relational expressions or logical expressions
&&	the conditional or short-circuit AND operator
||	the conditional or short-circuit OR operator

Short-Circuit Operator

```
if ( x || y) {
   statements;
}
```

For evaluating x || y:
First, evaluates x
    If x is true and then execute statements

    If x is false, then execute y.
        If y is true, execute statements



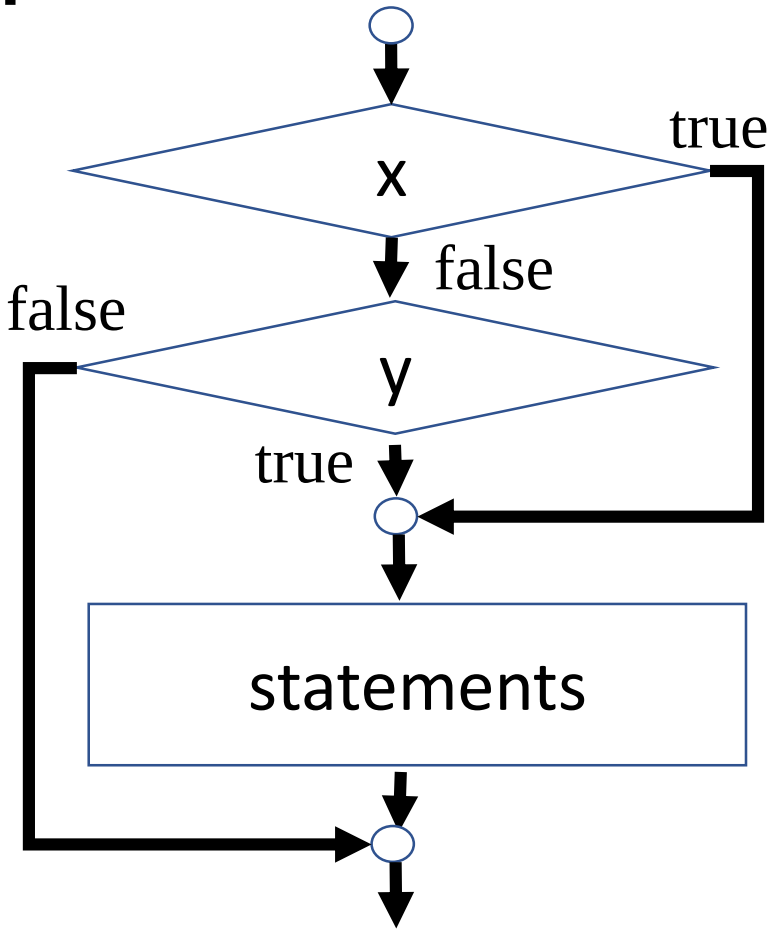
If x is true, y is not executed.
If x is false, y is executed.

Short-Circuit Operator

```
if ( x || y) {
   statements;
}
```

For evaluating x || y:
First, evaluates x
    If x is true and then execute statements

    If x is false, then execute y.
        If y is true, execute statements



If x is true, y is not executed.
If x is false, y is executed.

Example

Using and combining conditions

Example

Write a program to check whether a year is a leap year.

A leap year: either one of the conditions

I. it is divisible by 4 but not by 100

II. if it is divisible by 400.

Case I: (year % 4 == 0 && year % 100 != 0)

Case II: (year % 400 == 0)

Combine them:

(year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)

Example

Write a program to check whether a year is a leap year.

A leap year: either one of the conditions

I. it is divisible by 4 but not by 100

II. if it is divisible by 400.

Case I: (year % 4 == 0 && year % 100 != 0)

Case II: (year % 400 == 0)

Combine them:

(year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)

```
bool leapYear = false;
if (
    ( year % 4 == 0 && year % 100 != 0 )
    ||
    ( year % 400 == 0 )
) leapYear = true;
```

Example

Write a program to check whether a year is a leap year.

A leap year: either one of the conditions

I. it is divisible by 4 but not by 100

II. if it is divisible by 400.

Case I: (year % 4 == 0 && year % 100 != 0)

Case II: (year % 400 == 0)

Combine them:

(year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)

Shorten the lines

```
bool leapYear = false;
if (
    ( year % 4 == 0 && year % 100 != 0 )
    ||
    ( year % 400 == 0 )
) leapYear = true;
```

```
bool leapYear = false;
if (
    (    year % 4 == 0
      && year % 100 != 0
    )
    ||
    ( year % 400 == 0 )
) leapYear = true;
```

Intended Learning Outcomes

- Describe the processes of
  - sequence structures,
  - selection structures,
  - repetition structures,
  - short-circuits
- Describe the fall-through mechanism and *break* in a switch block

Supplemental Material

Operator Precedence

- var++, var--
- +, - (Unary plus and minus), ++var,--var
- (type) Casting
- ! (Not)
- *, /, % (Multiplication, division, and remainder)
- +, - (Binary addition and subtraction)
- <, <=, >, >= (Comparison)
- ==, !=; (Equality)
- && (Conditional AND) Short-circuit AND
- || (Conditional OR) Short-circuit OR
- ?: (right to left)
- =, +=, -=, *=, /=, %= (Assignment operator)

Common errors

if structures, assignment operator, and equality operator

Common Errors

Put down braces properly

```
if (side >= 0)
  square_area = side*side;
  cout     << "The area of the square is "
           <<  square_area
           << “.”
           << “Its side length is “
           << side << endl;
```

Equivalent to

```
if (side >= 0) {
  square_area = side*side;
}

  cout     << "The area of the square is "
           <<  square_area
           << “.”
           << “Its side length is “
           << side << endl;
```

Common Errors

Put down braces properly

```
if (side >= 0)
  square_area = side*side;
  cout     << "The area of the square is "
           <<  square_area
           << “.”
           << “Its side length is “
           << side << endl;
```

This line is inside the body of the if-structure.

Equivalent to

These lines are outside of the body of the if-structure.

These lines must be executed.

```
if (side >= 0) {
  square_area = side*side;
}

  cout     << "The area of the square is "
           <<  square_area
           << “.”
           << “Its side length is “
           << side << endl;
```

Common Errors

Put down braces properly

```
if (side >= 0)
  square_area = side*side;
  cout     << "The area of the square is "
           <<  square_area
           << “.”
           << “Its side length is “
           << side << endl;
```

```
if (side >= 0) {
  square_area = side*side;
}

  cout     << "The area of the square is "
           <<  square_area
           << “.”
           << “Its side length is “
           << side << endl;
```

Common Errors

Put down braces properly

```
if (side >= 0)
  square_area = side*side;
  cout     << "The area of the square is "
           <<  square_area
           << “.”
           << “Its side length is “
           << side << endl;
```

Good

```
if (side >= 0) {
  square_area = side*side;
  cout     << "The area of the square is "
           <<  square_area
           << “.”
           << “Its side length is “
           << side << endl;
}
```

Common Errors

Put down a semicolon in the correct place

```
if (side >= 0) ;
{ square_area = side*side;
    cout  << "The area of the square is "
          <<  square_area
          << “.”
          << “Its side length is “
          << side << endl;
}
```

```
if (side >= 0) { }
{ square_area = side*side;
    cout  << "The area of the square is "
          <<  square_area
          << “.”
          << “Its side length is “
          << side << endl;
}
```

Common Errors

Put down a semicolon in the correct place

```
if (side >= 0) ;
{ square_area = side*side;
    cout   << "The area of the square is "
           <<  square_area
           << “.”
           << “Its side length is “
           << side << endl;
}
```

Good

```
if (side >= 0) {
  square_area = side*side;
  cout     << "The area of the square is "
           <<  square_area
           << “.”
           << “Its side length is “
           << side << endl;
}
```

Common Errors

Put down a semicolon in the correct place

```
if (side >= 0) ; {
    square_area = side*side;
    cout  << "The area of the square is "
          <<  square_area
          << “.”
          << “Its side length is “
          << side << endl;
}
```

Good

```
if (side >= 0) {
  square_area = side*side;
  cout     << "The area of the square is "
           <<  square_area
           << “.”
           << “Its side length is “
           << side << endl;
}
```

Common Errors

Put down a semicolon in the correct place

```
if (side >= 0) ; {
    square_area = side*side;
    cout  << "The area of the square is "
          <<  square_area
          << “.”
          << “Its side length is “
          << side << endl;
}
```

Good

```
if (side >= 0) {
  square_area = side*side;
  cout     << "The area of the square is "
           <<  square_area
           << “.”
           << “Its side length is “
           << side << endl;
}
```

Common Errors

Put down a semicolon in the correct place

```
if (side >= 0) ; {
    square_area = side*side;
    cout  << "The area of the square is "
          <<  square_area
          << “.”
          << “Its side length is “
          << side << endl;
}
```

Good

```
if (side >= 0) {
  square_area = side*side;
  cout     << "The area of the square is "
           <<  square_area
           << “.”
           << “Its side length is “
           << side << endl;
}
```

Common Errors

= assignment operator

and

== equal operator

```
int x =0;
y = 0;

if (x=1) {
  y = 2;
}
```

Assign 1 to x.

Then check whether x is true or false.

Good

```
int x =0;
y = 0;

if ( x==1 ) {
   y = 2;
}
```

Check whether x is equal to 1.

Then check whether x is true or false.

Common Errors

= assignment operator

and

== equal operator

```
int x =0;
y = 0;

if (x=1+y*9) {
  y = 2;
}
```

Assign (1+y*9) to x.

Then check whether x is true or false.

Good

```
int x =0;
y = 0;

if ( x==1 + y*9 ) {
   y = 2;
}
```

Check whether x is equal to (1 + y*9).

Then check whether x is true or false.

Relational Operators

Operator	Name	Example	Result
<	less than	2 < 3	true
<=	less than or equal to	2 <= 3	true
>	greater than	2 > 3	false
>=	greater than or equal to	2 >= 3	false
==	equal to	2 == 3	false
!=	not equal to	2 != 3	true

Input using cin

Using an if-structure to check for a valid input

```
#define PI 3.141592654 // define a preprocessor macro
void foo( ) {
    cin >> radius;
    if ( radius < 0 ) return;
    area = radius*radius*PI;
}
```

Operators

How to evaluate the following expression?

$$3 + 4 * 4 > 5 * (4 + 3) - 1?$$

What is the result?

= and others

```
int a = 6, b = 5, c = 3;
```

a = b = a * c - 1 > 5 ? 3 > 1 + a + c == 1 > 1: 5+b;

What are the values of a, b, and c after the expression is executed?

=, and others

```
int a = 6, b = 5, c = 3;
```

a = b*c, b = a + 1, c = a*5 - b;

Left to right

What are the values of a, b, and c after the expression is executed?

for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0, j = 1;
    ;
    ++i ) {
    if ( i > 20) break;
    s += 2 + j;
    j = j + 2;
}
```

What is the value of s after the fragment is executed?

```
s = 0;
for (int i = 0, j = 1 ;; ++i ) {
    if ( i > 20) break;
    s += 2 + j;
    j = j + 2;
}
```

What is the value of s after the fragment is executed?

Infinite loop

for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

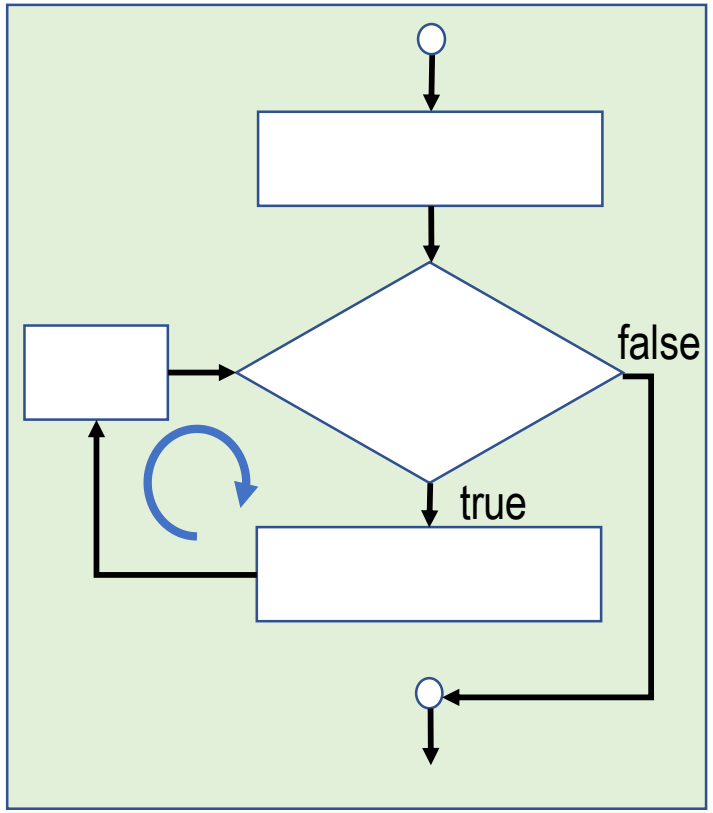
Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0, j = 1;
      ;
      ++i ) {
      if ( i > 20) break;
      s += 2 + j;
      j = j + 2;
}
```

What is the value of s after the fragment is executed?



for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

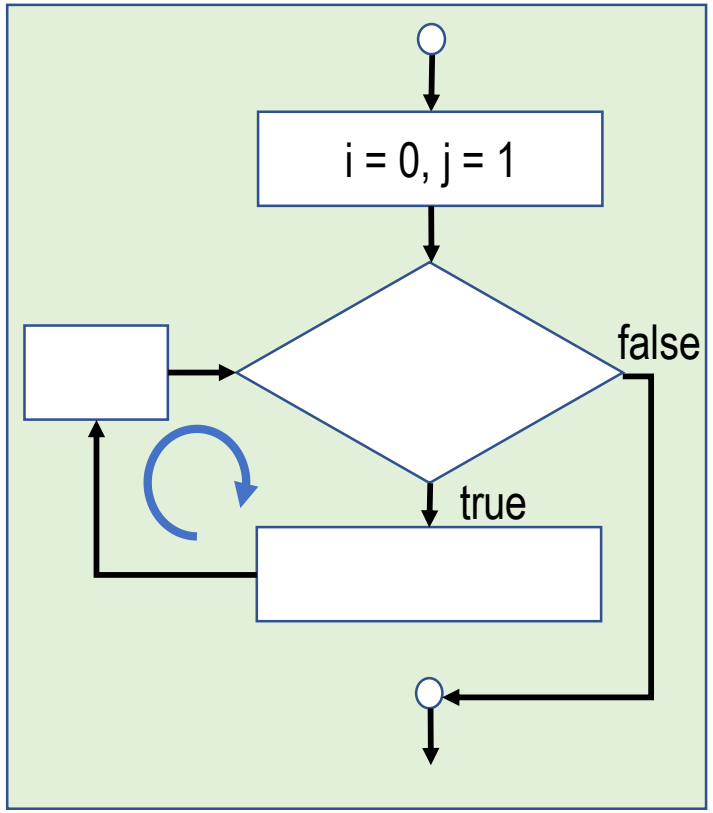
Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0, j = 1;
      ;
      ++i ) {
      if ( i > 20) break;
      s += 2 + j;
      j = j + 2;
}
```

What is the value of s after the fragment is executed?



for-loop

```
for (Exp1; Exp2; Exp3) {
        body
}
```

Exp1: initialization expression

Exp2: condition expression

Exp3: counter update expression

1. Execute Exp1
2. Evaluate Exp2
3. If Exp2 is true, execute body
4. If Exp2 is false, exit the structure
4. Execute Exp3
5. Go to step 2

```
s = 0;
for (int i = 0, j = 1;
    ;
    ++i ) {
    if ( i > 20) break;
    s += 2 + j;
    j = j + 2;
}
```

What is the value of s after the fragment is executed?