Section Overview

Controlling Program Flow

- Sequence
  - Ordering statements sequentially
- Selection
  - Making decisions
- Iteration
  - Looping or repeating

Selection – Decision Making

- `if` statement
- `if-else` statement
- Nested `if` statements
- `switch` statement
- Conditional operator `?:`

Iteration - Looping

- `for` loop
- Range-based `for` loop
- `while` loop
- `do-while` loop
- `continue` and `break`
- Infinite `loops`
- Nested `loops`

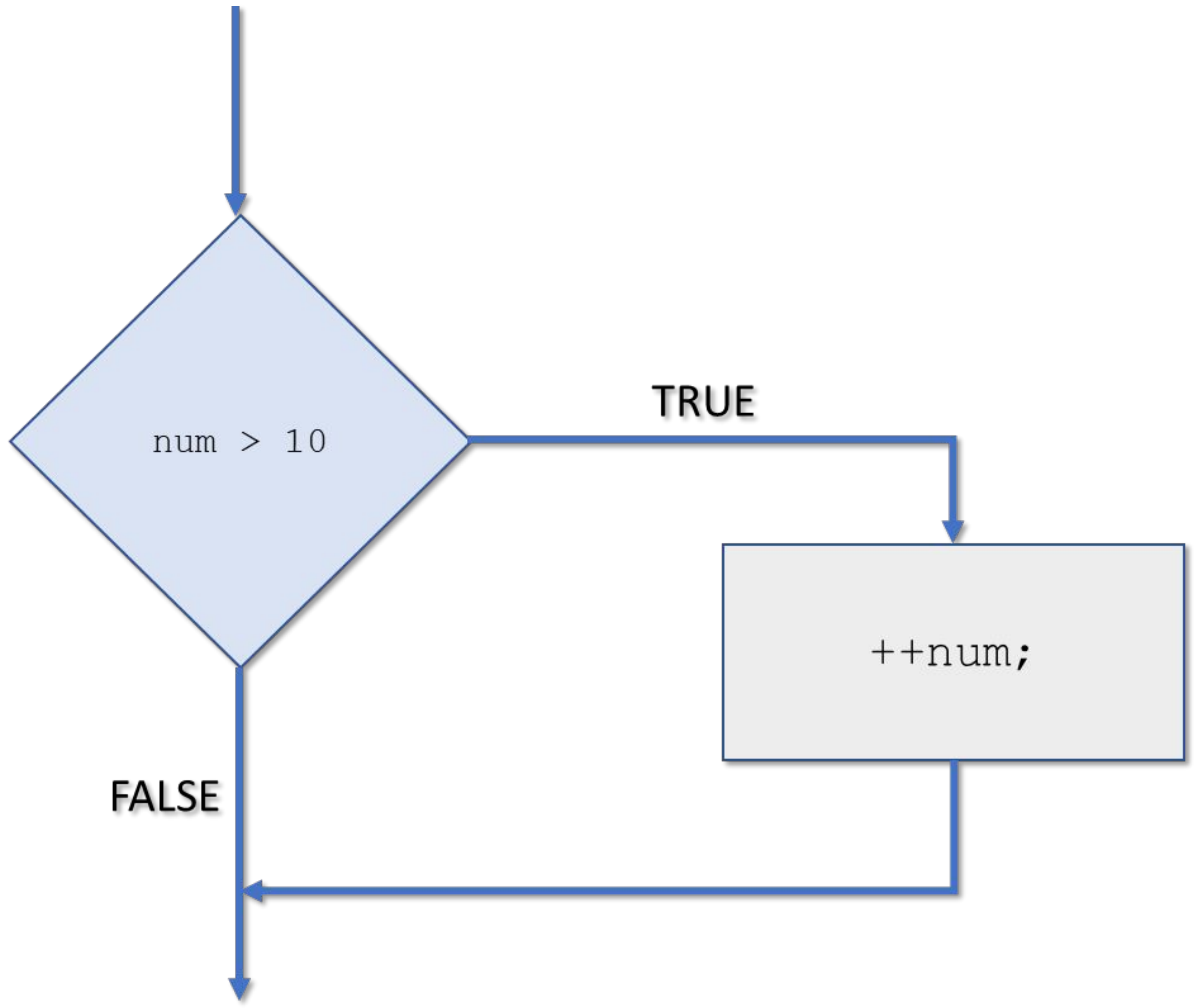
`if` statement

```
if (expr)
  statement;
```

- If the expression is true then execute the statement
- If the expression is false then skip the statement

`if` statement

```
if(num > 10)
   ++num;
```



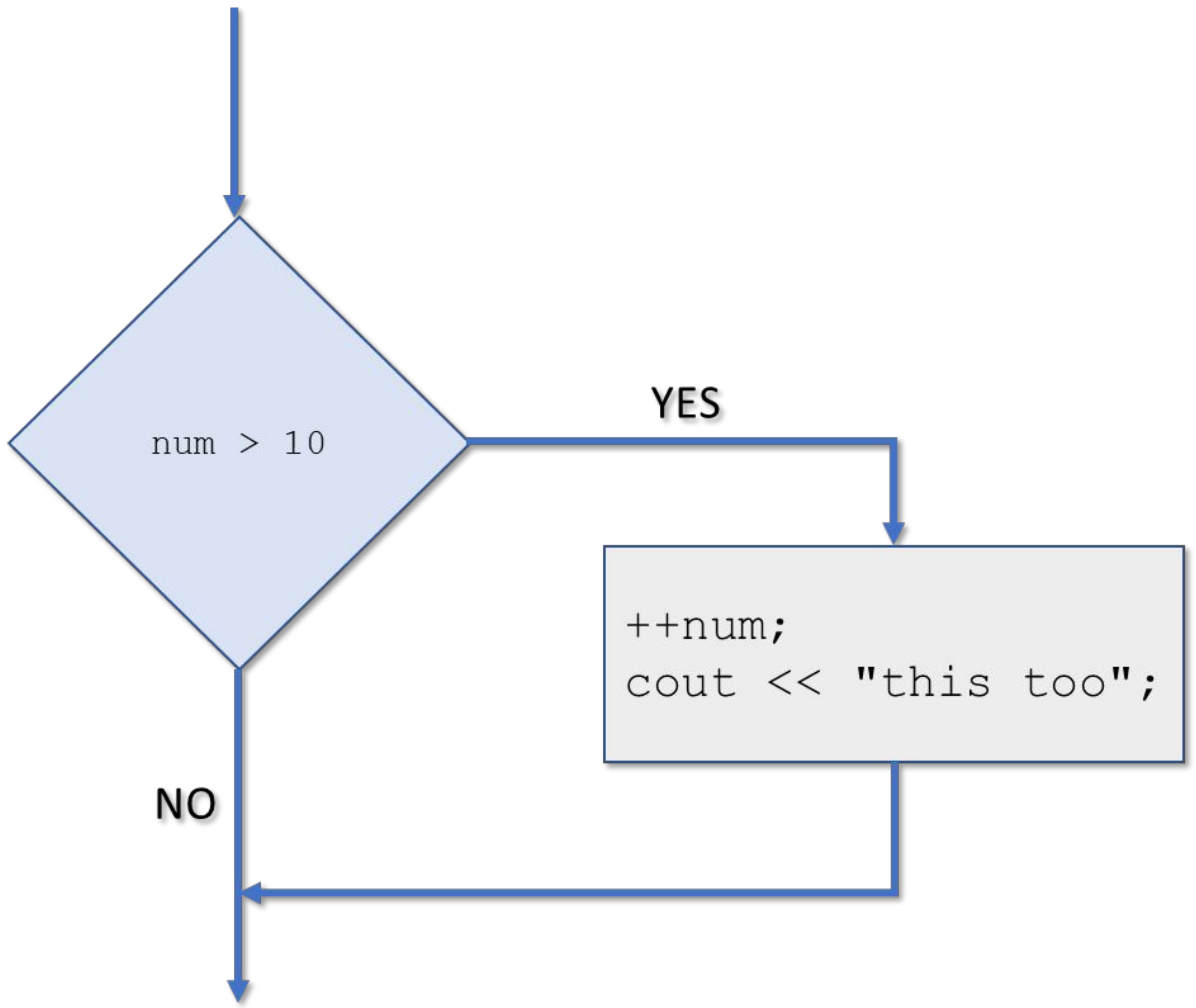
`if` statement

```
if (selection == 'A')
   cout << "You selected A";

if (num > 10)
   cout << "num is greater than 10";

if (health < 100 && player_healed)
   health = 100;
```

`if` statement

block statement

```
if(num > 10){
   ++num;
   cout << "this too";
}
```



Block statement

```
{
    //variable declarations
    statement1;
    statement2;
    . . .
}
```

- Create a block of code by including more than one statement in code block { }
- Blocks can also contain variable declarations
- These variables are visible only within the block – local scope

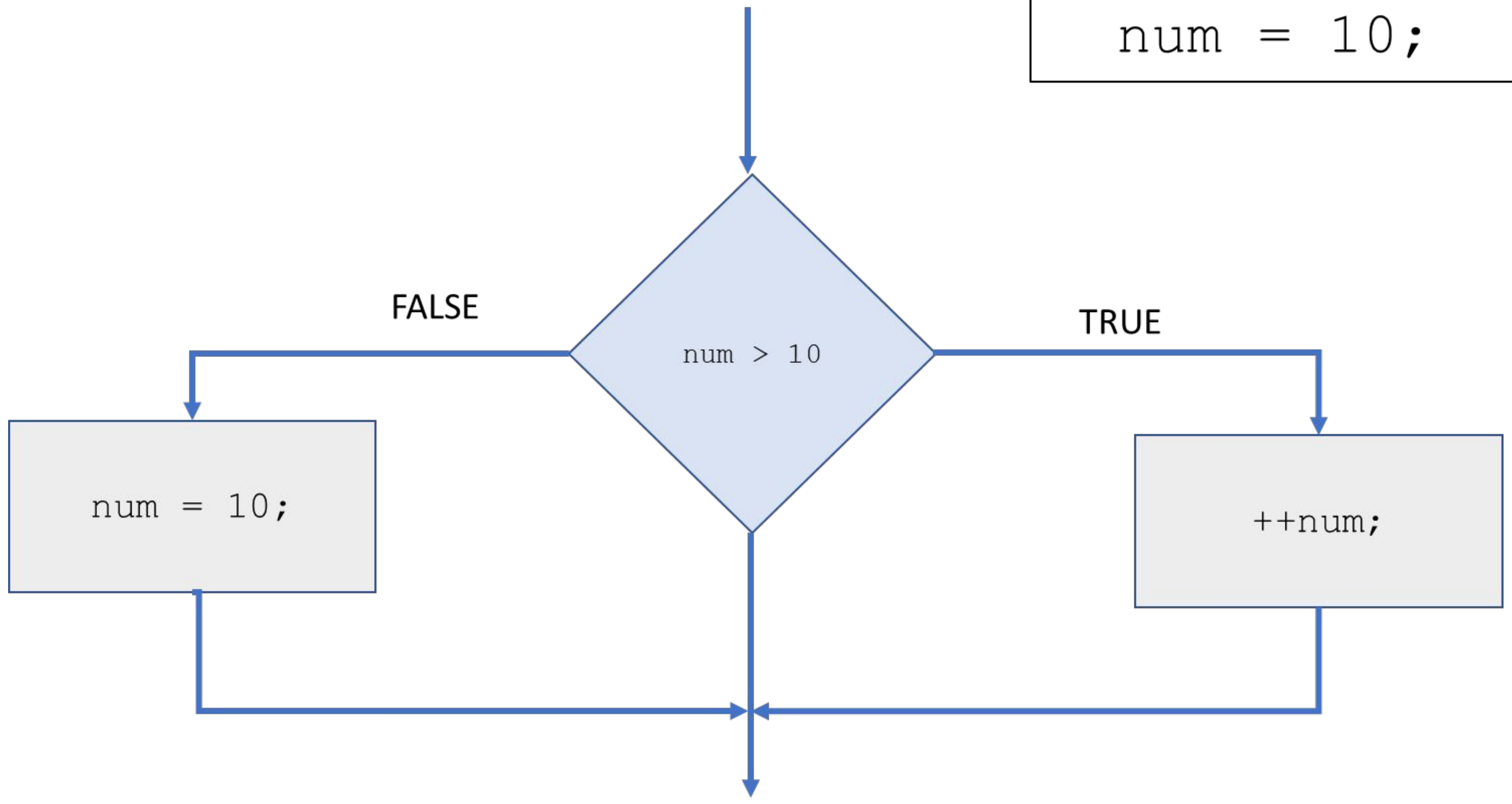
`if-else` statement

```
if  (expr)
    statement1;
else
    statement2;
```

- If the expression is **true** then execute **statement1**
- If the expression is **false** then execute **statement2**

`if-else` statement

```
if (num > 10)
   ++num;
else
   num = 10;
```



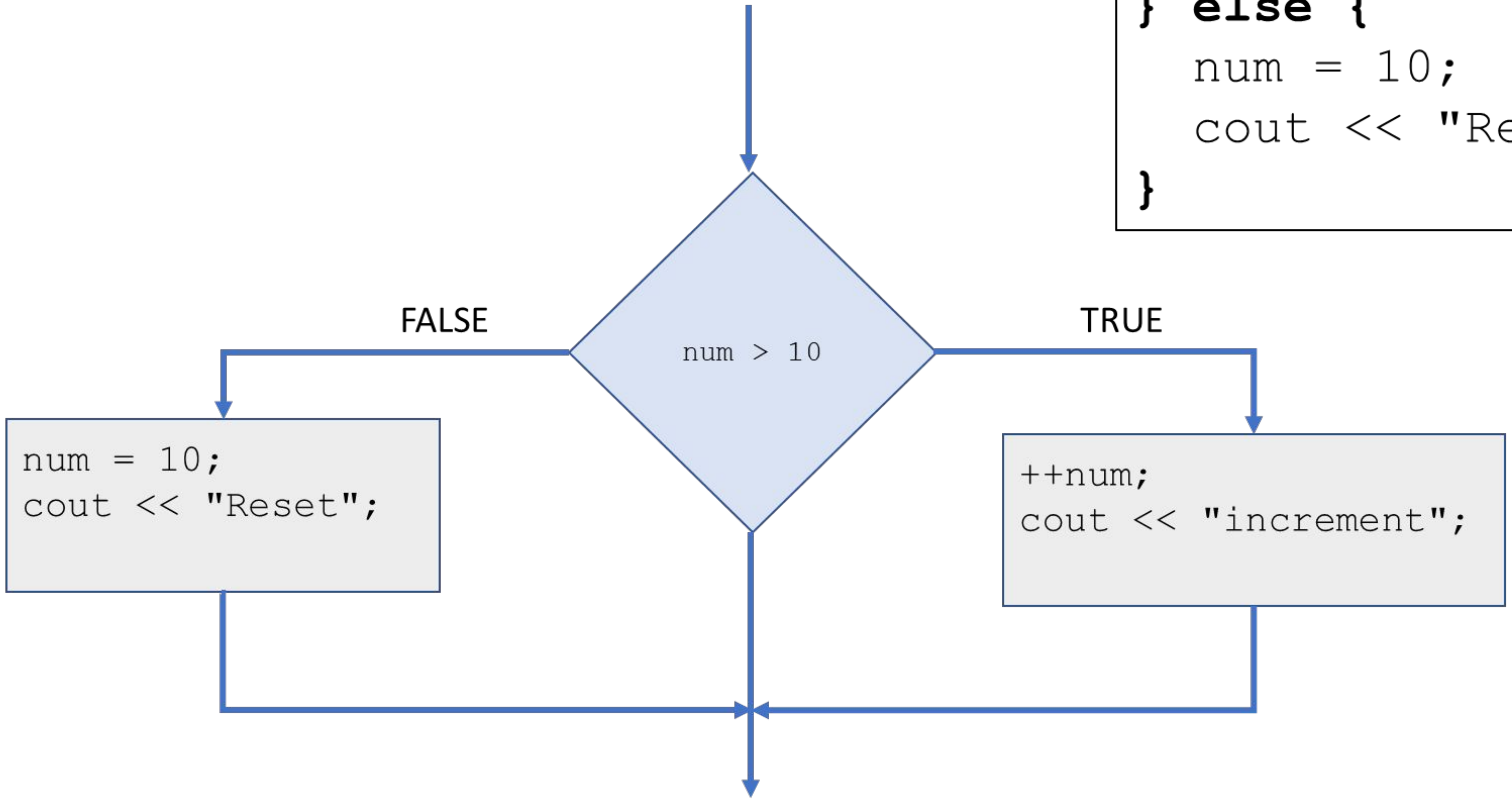
`if-else` statement

```
if (num > 10)
    cout << "num is greater than 10";
else
    cout << "num is NOT greater than 10";



if (health < 100 && heal_player)
    health = 100;
else
    ++health;
```

`if-else` statement

block statement

```
if(num > 10){
   ++num;
   cout << "increment";
} else {
   num = 10;
   cout << "Reset";
}
```



`if-else-if` construct

block statement

```
if (score > 90)
    cout <<  "A";
else if (score > 80)
    cout <<  "B";

else if (score > 70)
    cout <<  "C";

else if (score > 60)
    cout <<  "D";

else                          // all others must be F
    cout << "F";


cout << "Done";
```

Nested `if` statement

```
if  (expr1)
    if  (expr2)
        statement1;
    else
        statement2;
```

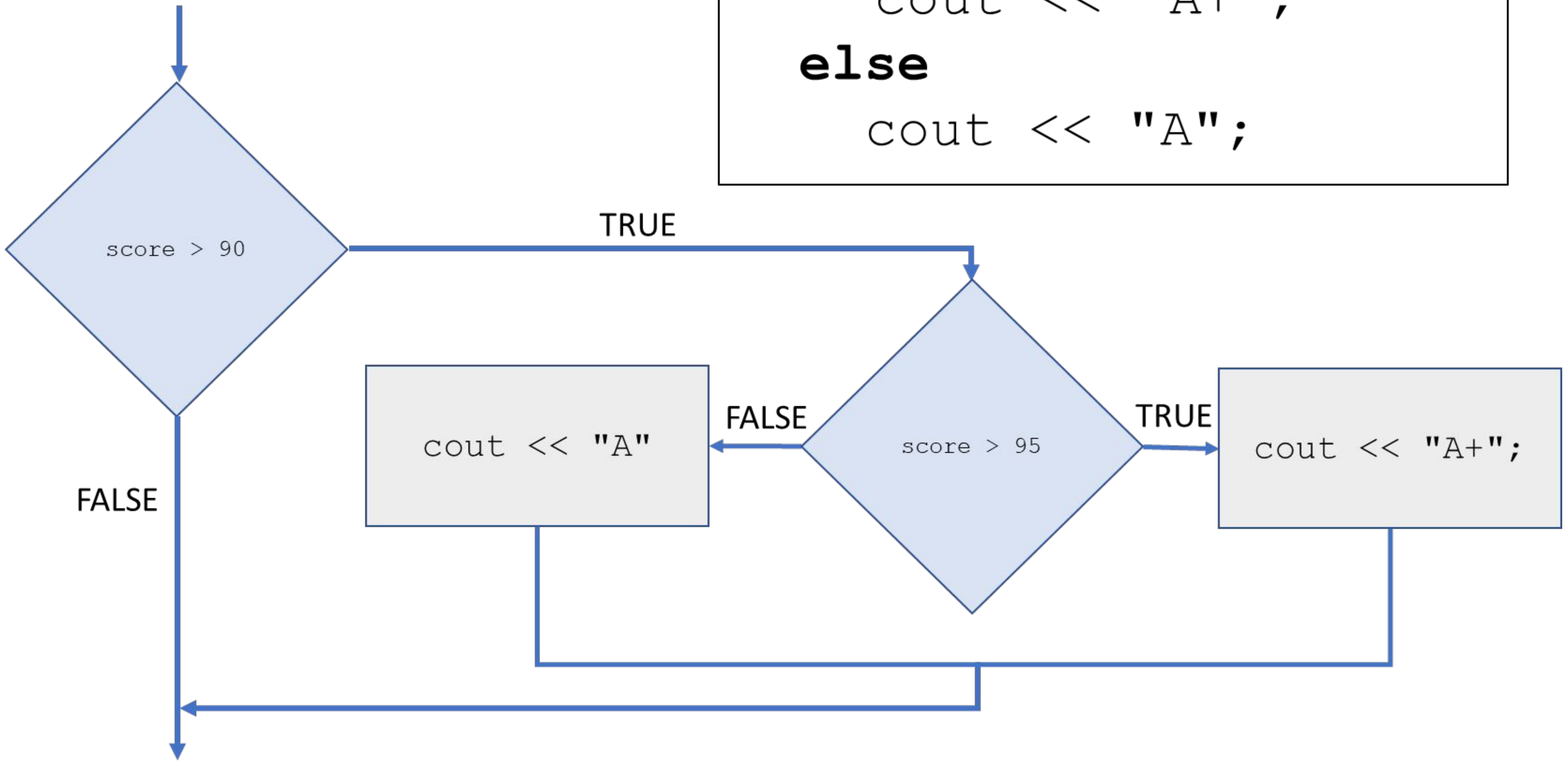
- **`if`** statement is nested within another
- Allows testing of multiple conditions
- **`else`** belongs to the closest **`if`**

Nested `if` statement

```
if (score > 90)
  if (score > 95)
     cout << "A+";
  else
     cout << "A";

else
  cout << "Sorry, No A";
```

Nested `if` statement

```
if (score > 90)
  if (score > 95)
    cout << "A+";
  else
    cout << "A";
```



Nested `if` statement

```
if (score_frank != score_bill) {
    if (score_frank > score_bill) {
        cout << "Frank Wins" << endl;
    } else {
        cout << "Bill Wins" << endl;
    }
} else {
    cout << "Looks like a tie!" << endl;
}
```

The `switch` statement

```
switch (integer_control_expr) {
    case expression_1: statement_1; break;
    case expression_2: statement_2; break;
    case expression_3: statement_3; break;
     . . .
    case expression_n: statement_n; break;
    default: statement_default;
}
```

The `switch` statement

example

```
switch (selection) {
    case '1': cout << "1 selected";
              break;
    case '2': cout << "2 selected";
              break;
    case '3':
    case '4': cout << "3 or 4 selected";
              break;
    default:  cout << "1,2,3,4 NOT selected";
}
```

The `switch` statement

fall-through example

```
switch (selection) {
    case '1': cout << "1 selected";

    case '2': cout << "2 selected";

    case '3': cout << "3 selected";

    case '4': cout << "4 selected";
              break;
    default:  cout << "1,2,3,4 NOT selected";
}
```

The `switch` statement

with an enumeration

```
enum Color {
    red, green, blue
};
Color screen_color {green};
```

```
switch (screen_color) {
   case red:    cout << "red"; break;
   case green:  cout << "green"; break;
   case blue:   cout << "blue"; break;
   default:     cout << "should never execute";
}
```

The `switch` statement

- The control expression must evaluate to an integer type

- The case expressions must be constant expressions that evaluate to integer or integers literals

- Once a match occurs all following case sections are executes UNTIL a `break` is reached the switch complete

- Best practice – provide break statement for each case
- Best practice – `default` is optional, but should be handled

Conditional Operator

?:

```
(cond_expr) ? expr1 : expr2
```

- cond_expr evaluates to a boolean expression
    - If cond_expr is true then the value of expr1 is returned
    - If cond_expr is false then the value of expr2 is returned
- Similar to if-else construct
- Ternary operator
- Very useful when used inline
- Very easy to abuse!

Conditional Operator

example

```
int a{10}, b{20};
int score{92};
int result {};

result = (a > b) ? a : b;

result = (a < b) ? (b-a) : (a-b);

result = (b != 0) ? (a/b) : 0;

cout << ((score > 90) ? "Excellent" : "Good ");
```