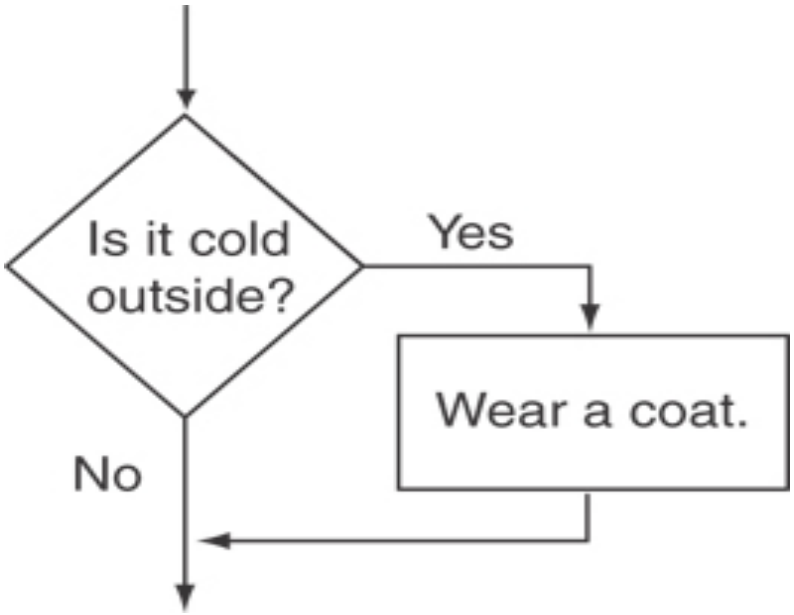
REVIEW

SELECTION

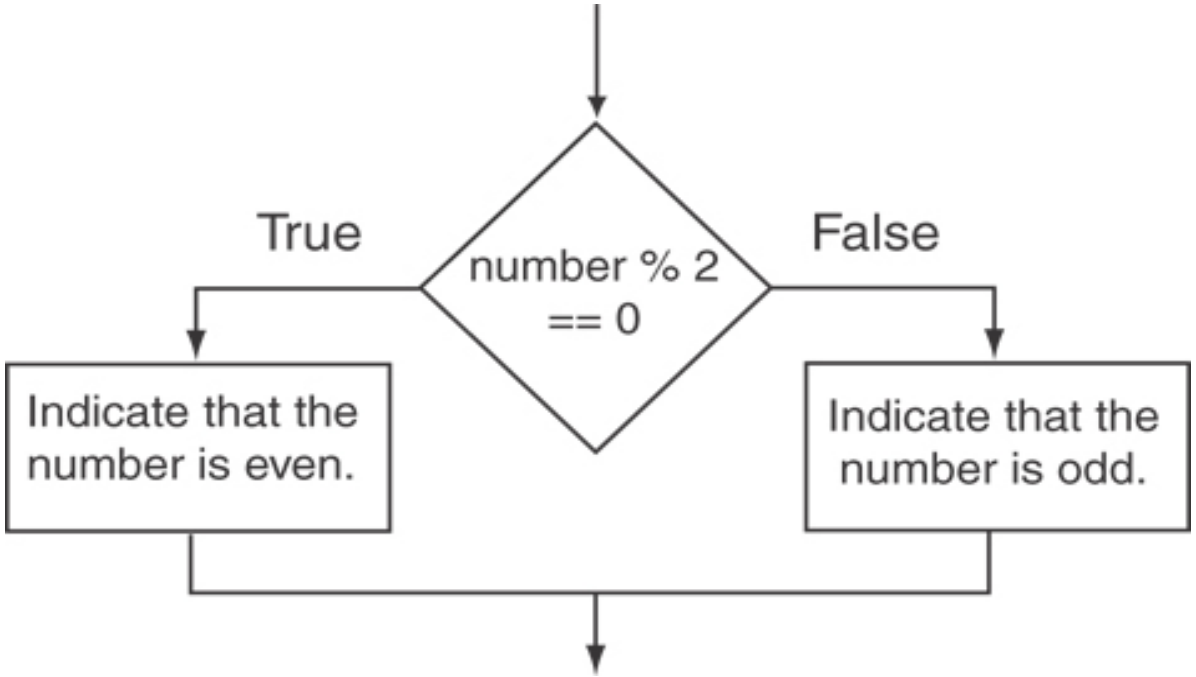
1-way Selection

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



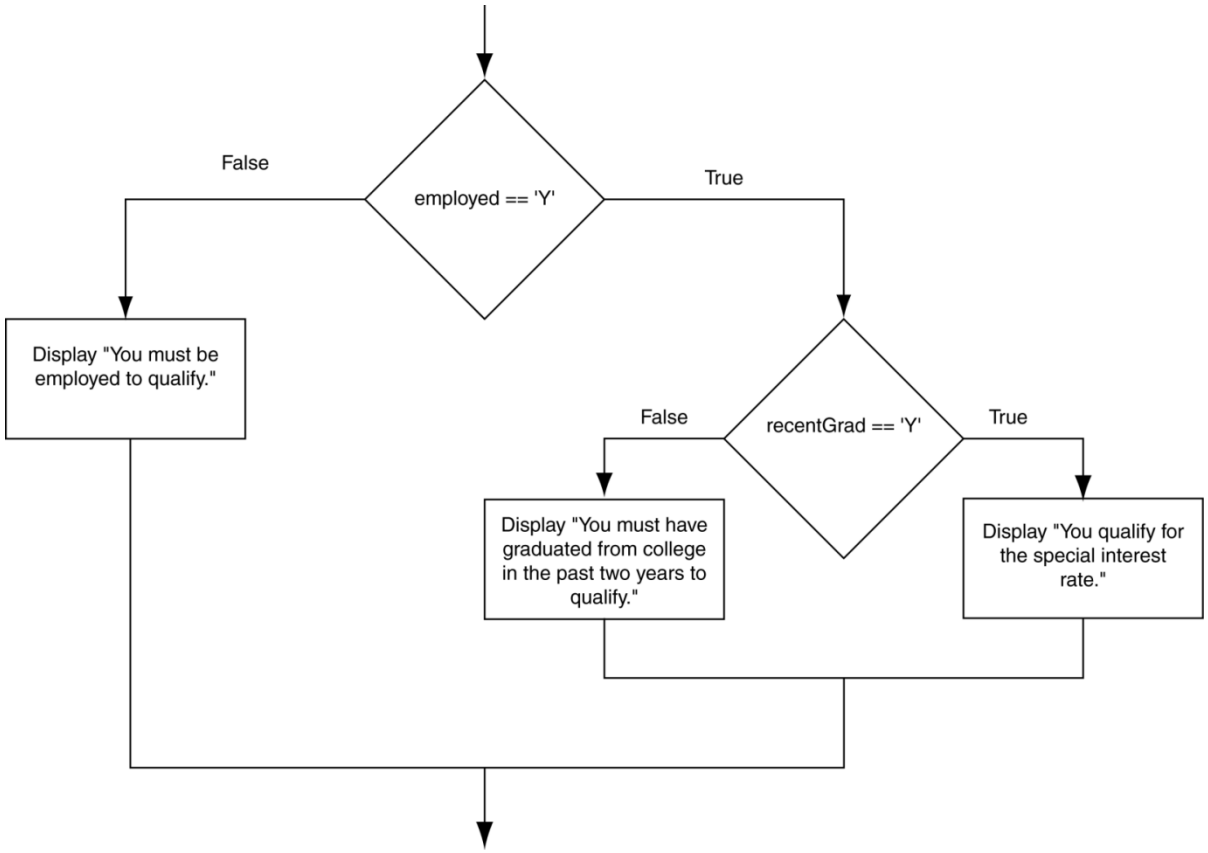
2-way Selection

```
if (expression)
        statement1;   // or block
 else
        statement2;   // or block
```



Multiway Selection (1 of 2)

- Nested **`if`** Statements



- **`if/else if`** Statement

Multiway Selection (2 of 2)

- **`switch`** Statement

```
switch (expression) //integer
{
  case exp1: statement1; break;
  case exp2: statement2; break;
  ...
  case expn: statementn; break;
  default:   statementn+1;
}
```

The Conditional Operator

- Can use to create short `if/else` statements
- Format: `expr ? expr : expr;`
- Ex.

```
max =(num1 >num2) ? num1 : num2;
```

```
x<0 ? y=10 : z=20;
```

First Expression: Expression to be tested

2nd Expression: Executes if first expression is true

3rd Expression: Executes if the first expression is false

REVIEW

REPETITION

LOOP CONTROL STRUCTURE

- Loop: a control structure that causes a statement or statements to repeat
- Pretest

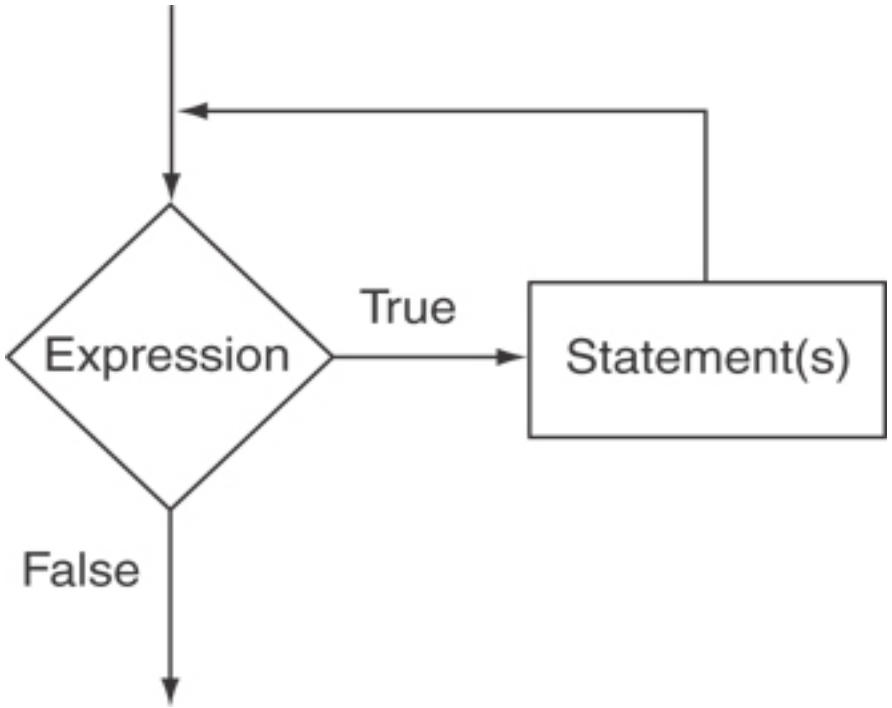
  `while`

  `for`
- Post-test

  `do/while`
- Nested loops
  - loop within a loop

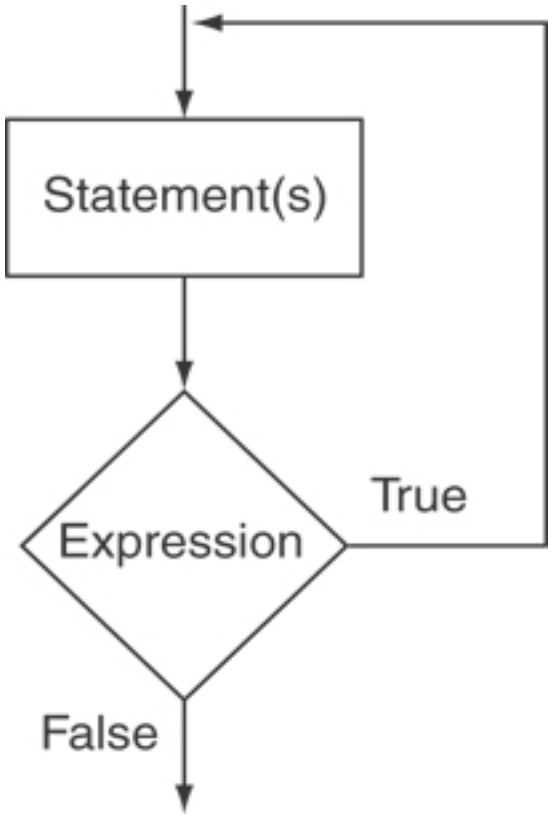
while Loop

```
while (expression)
        statement;
```



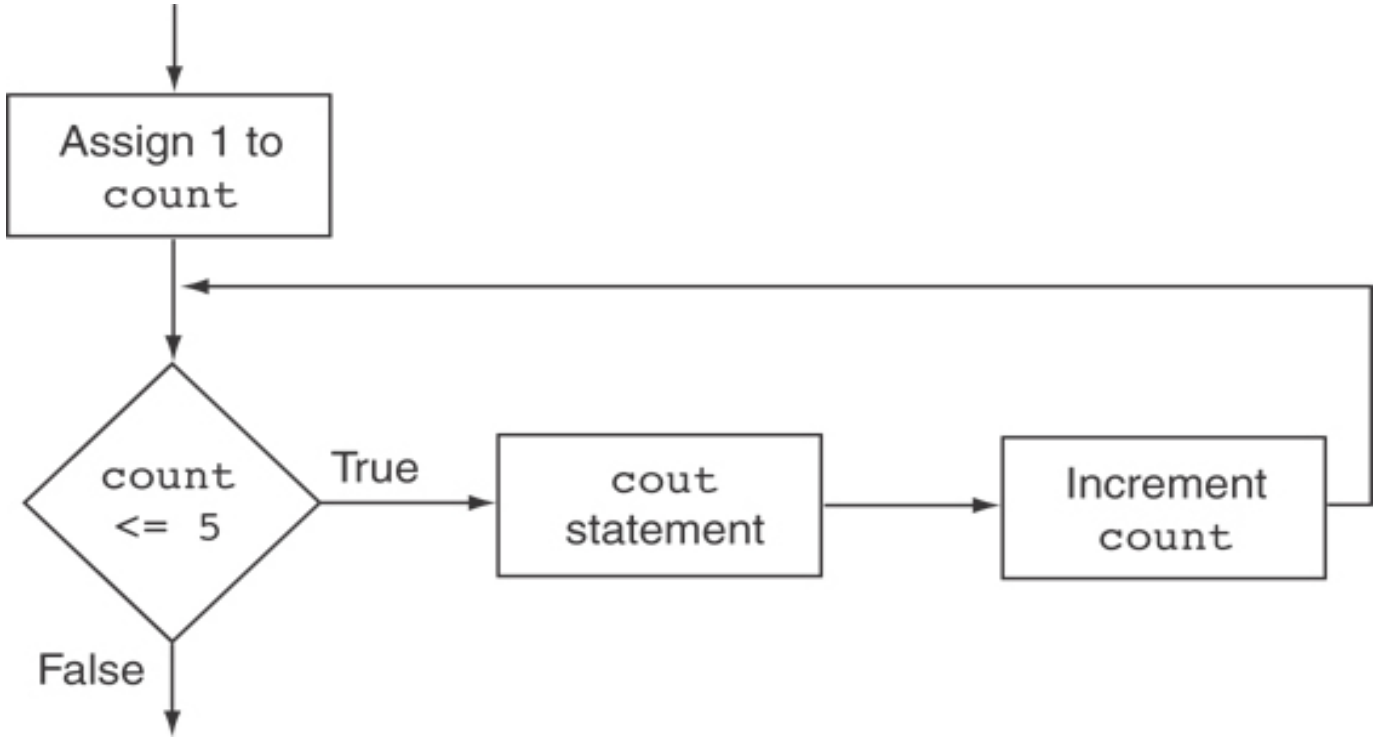
do-while Loop

```
do
    statement;  // or block in { }
 while (expression);
```



for Loop

```
for(initialization; test; update)
        statement; // or block in { }
```



Counter-controlled

- Counter: a variable that is incremented or decremented each time a loop repeats
- Can be used to control execution of the loop (also known as the *loop control variable*)
- Must be initialized before entering loop

Sentinel-controlled

- <u>sentinel</u>: value in a list of values that indicates end of data
- Special value that cannot be confused with a valid value, *e.g.*, `-999` for a test score
- Used to terminate input when user may not know how many values will be entered

Flag-controlled

- flag: a boolean value
- Set to either TRUE or FALSE as initializers when loop starts
- Value will be updated to the opposite of the initialised value twhen it meets a loop-ending condition

`break` Statement

- Can use `break` to terminate execution of a loop
- Use sparingly if at all – makes code harder to understand and debug
- When used in an inner loop, terminates that loop only and goes back to outer loop

`continue` Statement

- Can use `continue` to go to end of loop and prepare for next repetition
  - `while`, `do-while` loops: go to test, repeat loop if test passes
  - `for` loop: perform update step, then test, then repeat loop if test passes
- Use sparingly – like `break`, can make program logic hard to follow