CIS 22A – Lecture 7

Manish Goel

Multiple Assignments

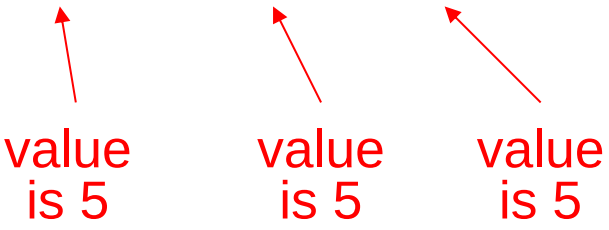
- The = can be used to assign a value to multiple variables:

```
x = y = z = 5;
```

- Value of = is the value that is assigned

- Associates right to left:

```
x = (y = (z = 5));
```



Combined Assignment

- `sum = sum + 1;` // adds 1 to the variable **`sum`**
- `res = res * 2;` // multiplies variable **`res`** by 2

- Combined assignment operators provide shorthand for such operations.
- `sum = sum + 1;` ➔ `sum += 1;`
- `Res = res * 2;` ➔ `res *= 2;`

- For addition and subtraction, shorthand for incrementing / decrementing
- `x = x + 1` ➔ `x += 1` ➔ `x++`
- `y = y - 1` ➔ `x -= 1` ➔ `y--`

Table 3-9

Operator	Example Usage	Equivalent to
+=	`x += 5;`	`x = x + 5;`
-=	`y -= 2;`	`y = y - 2;`
*=	`z *= 10;`	`z = z * 10;`
/=	`a /= b;`	`a = a / b;`
%=	`c %= 3;`	`c = c % 3;`

Relational Operators & Expressions

Table 4-1

Relational Operators	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
! =	Not equal to

All of the relational operators are binary, which means they use two operands. Here is an example of an expression using the greater-than operator:

Table 4-2

Expression	What the Expression Means
`x > y`	Is x greater than y?
`x < y`	Is x less than y?
`x >= y`	Is x greater than or equal to y?
`x <= y`	Is x less than or equal to y?
`x == y`	Is x equal to y?
`x ! = y`	Is x not equal to y?

NOTE: All the relational operators have left-to-right associativity. Recall that associativity is the order in which an operator works with its operands.

Value of Relational Expressions

Table 4-3 (Assume x is 10 and y is 7.)

Expression	Value
`x < y`	False, because x is not less than y.
`x > y`	True, because x is greater than y.
`x >= y`	True, because x is greater than or equal to y.
`x <= y`	False, because x is not less than or equal to y.
`y != x`	True, because y is not equal to x.

NOTE: Relational expressions have a higher precedence than the assignment operator. In the statement

```
z = x < y;
```

the expression `x < y` is evaluated first, and then its value is assigned to z.

Table 4-4 (Assume x is 10, y is 7, and z, a, and b are `int`s or `bool`s)

Statement	Outcome
`z = x < y`	z is assigned 0 because x is not less than y.
`cout << (x > y);`	Displays 1 because x is greater than y.
`a = x >= y;`	a is assigned 1 because x is greater than or equal to y.
`cout << (x <= y);`	Displays 0 because x is not less than or equal to y.
`b = y != x;`	b is assigned 1 because y is not equal to x.

Things to remember - relational expressions

The Value of a Relationship

- When a relational expression is true it has the value 1.
- When a relational expression is false it has the value 0.
- Any expression that has the value 0 is considered false by the `if` statement. This includes the `bool` value `false`, which is equivalent to 0.
- Any expression that has any value other than 0 is considered true by the `if` statement. This includes the `bool` value `true`, which is equivalent to 1.

Don't Confuse == With =

Earlier you saw a warning not to confuse the equality operator (==) with the assignment operator (=) , as in the following statement:

```
if (x = 2)  //Caution here!
   cout << "It is True!";
```

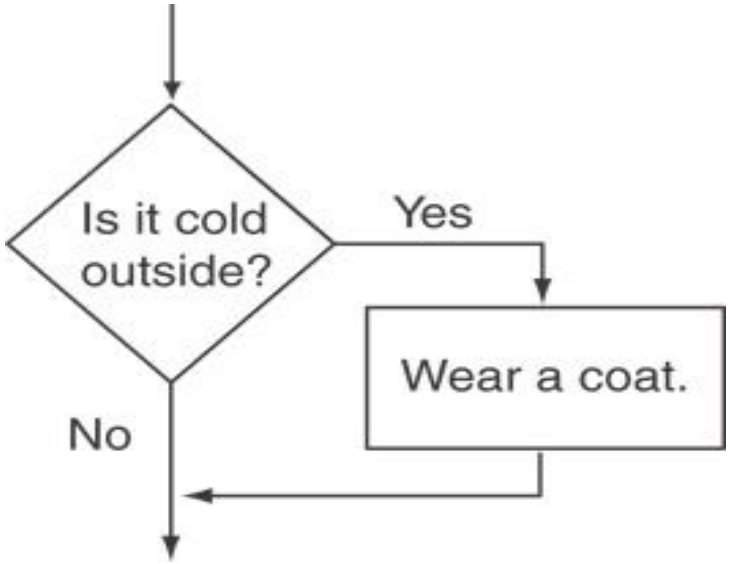
The statement above does not determine whether x is equal to 2, it assigns x the value 2! Furthermore, the cout statement will *always* be executed because the expression x = 2 is always true.

Comparing Floating-Point Numbers

Because of the way that floating-point numbers are stored in memory, rounding errors sometimes occur. This is because some fractional numbers cannot be exactly represented using binary. So, you should be careful when using the equality operator (==) to compare floating point numbers. For example, Program 4-4 uses two `double` variables, a and b.

Conditional Programming – the `if` statement

- Allows statements to be conditionally executed or skipped over
- Models human evaluation process – but differently
  - Is it cold outside? Do I wear a jacket? : human evaluation
  - "If it is cold outside, wear a jacket : programming
  - Flowchart for the condition



The `if` statement – What Happens

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

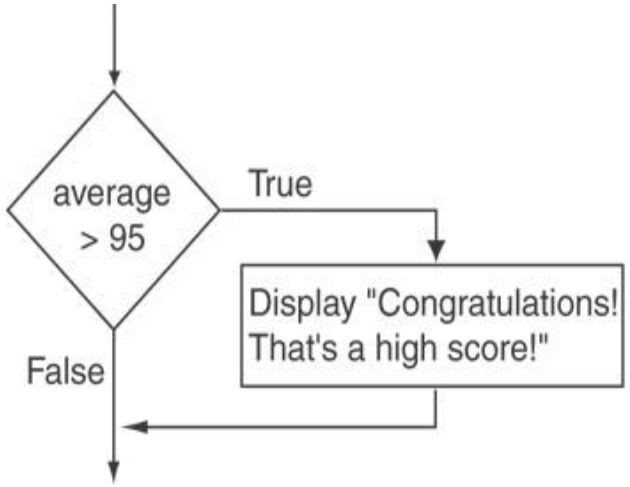
- If *`expression`* is `true`, then *`statement`* is executed.
- If *`expression`* is `false`, then *`statement`* is skipped.
- Do not place "`;`" after (*`expression`*)
- Place "*`statement`*`;`" on a separate line after (*`expression`*), indented:

```
if (score > 90)
    grade = 'A';
```

- Be careful testing `float`s and `double`s for equality
- `0` is `false`; any other value is `true`

Program 4-2

```
1  // This program averages three test scores
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  int main()
7  {
8     int score1, score2, score3;  // To hold three test scores
9     double average;              // To hold the average score
10
11    // Get the three test scores.
12    cout << "Enter 3 test scores and I will average them: ";
13    cin >> score1 >> score2 >> score3;
14
15    // Calculate and display the average score.
16    average = (score1 + score2 + score3) / 3.0;
17    cout << fixed << showpoint << setprecision(1);
18    cout << "Your average is " << average << endl;
19
20    // If the average is greater than 95, congratulate the user.
21    if (average > 95)
22       cout << "Congratulations! That's a high score!\n";
23    return 0;
24 }
```



Program Output with Example Input Shown in Bold

```
Enter 3 test scores and I will average them: 80 90 70 [Enter]
Your average is 80.0
```

Program Output with Other Example Input Shown in Bold

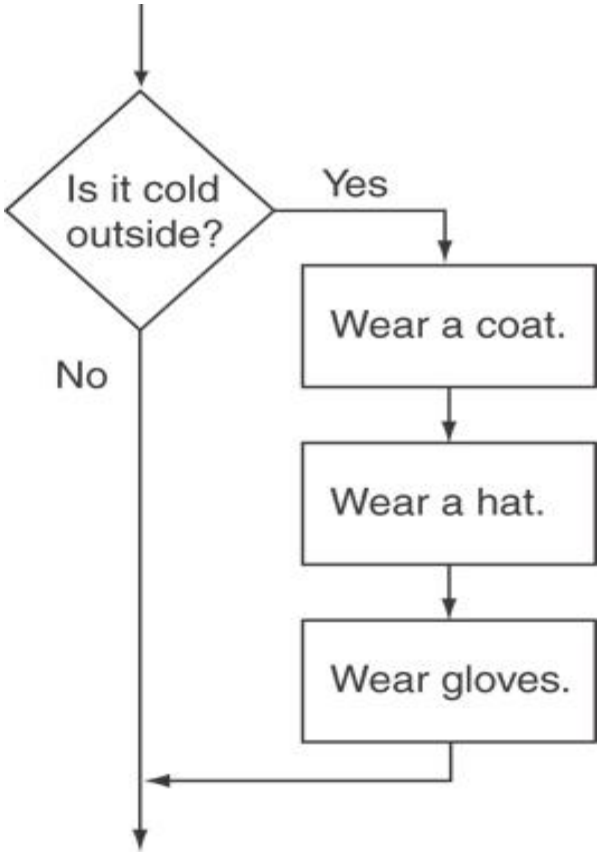
```
Enter 3 test scores and I will average them: 100 100 100 [Enter]
Your average is 100.0
Congratulations! That's a high score!
```

Expanding the `if` statement

- To execute more than one statement as part of an `if` statement, enclose them in braces (`{ }`):

```
if (score > 90)
{
   grade = 'A';
   cout << "Good Job!\n";
 }
```

- `{ }` creates a block of code



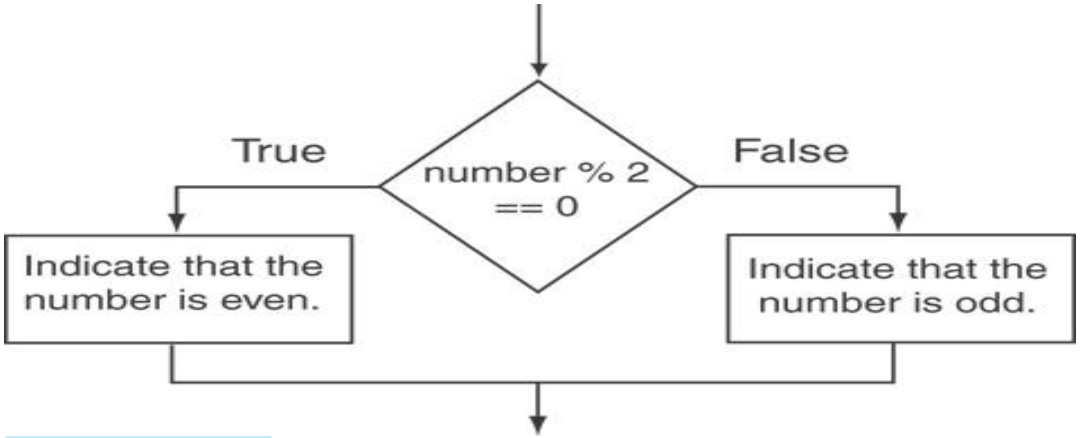
`if/else` – one or the other

- Provides two possible paths of execution
- Performs one statement/block if the *`expression`* is true, otherwise performs another statement/block.
- General Format:

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

- If the *`expression`* is `true`, then *`statement1`* is executed and *`statement2`* is skipped.
- If the *`expression`* is *`false`*, then *`statement1`* is skipped and *`statement2`* is executed.

To test if a number is odd or even




```
1  // This program uses the modulus operator to determine
2  // if a number is odd or even. If the number is evenly divisible
3  // by 2, it is an even number. A remainder indicates it is odd.
4  #include <iostream>
5  using namespace std;
6
7  int main()
8  {
9     int number;
10
11    cout << "Enter an integer and I will tell you if it\n";
12    cout << "is odd or even. ";
13    cin >> number;
14    if (number % 2 == 0)
15       cout << number << " is even.\n";
16    else
17       cout << number << " is odd.\n";
18    return 0;
19 }
```

Program Output with Example Input Shown in Bold

```
Enter an integer and I will tell you if it
is odd or even. 17 [Enter]
17 is odd.
```

`if/else if` – more than one condition

- Tests a series of conditions until one is found to be true

```
if (expression)
        statement1;  // or block
else if (expression)
        statement2;  // or block
       .
       . // other else ifs
       .
else if (expression)
        statementn-1;  // or block
else  (expression)
        statementn;  // or block
```

- The trailing `else` clause is optional, but it is best used to catch errors.

`switch` – alternate to `if/else if`

- Can be used to select among statements from several alternatives instead of `if/else if`

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

`switch` - Requirements

1) *`expression`* is evaluated - must be an integer variable or an expression that evaluates to an integer value

2) *`exp`**`1`*** through *`exp`**`n`*** must be constant integer expressions or literals, and must be unique in the `switch` statement

3) The value of *`expression`* is compared against *`exp`**`1`*** through *`exp`**`n`***. If *`expression`* matches value *`exp`**`i`***, the program branches to the statement following *`exp`**`i`*** and continues to the end of the `switch`

4) `default` is optional but recommended - If no matching value is found, the program branches to the statement after `default:`

5) `break` is used to exit a `switch` statement - if left out, the program "falls through" the remaining statements in the `switch` statement

The **`switch`** Statement in Program 4-23

Program 4-23

```
1  // The switch statement in this program tells the user something
2  // he or she already knows: the data just entered!
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     char choice;
9
10    cout << "Enter A, B, or C: ";
11    cin >> choice;
12    switch (choice)
13    {
14       case 'A': cout << "You entered A.\n";
15                 break;
16       case 'B': cout << "You entered B.\n";
17                 break;
18       case 'C': cout << "You entered C.\n";
19                 break;
20       default:  cout << "You did not enter A, B, or C!\n";
21    }
22    return 0;
23 }
```

Program Output with Example Input Shown in Bold

```
Enter A, B, or C: B [Enter]
You entered B.
```

Program Output with Example Input Shown in Bold

```
Enter A, B, or C: F [Enter]
You did not enter A, B, or C!
```

break and **default** statements in Pgm 4-25

Program 4-25

```
1  // This program is carefully constructed to use the "fall through"
2  // feature of the switch statement.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     int modelNum;  // Model number
9
10    // Get a model number from the user.
11    cout << "Our TVs come in three models:\n";
12    cout << "The 100, 200, and 300. Which do you want? ";
13    cin >> modelNum;
14
15    // Display the model's features.
16    cout << "That model has the following features:\n";
17    switch (modelNum)
18    {
19       case 300: cout << "\tPicture-in-a-picture.\n";
20       case 200: cout << "\tStereo sound.\n";
21       case 100: cout << "\tRemote control.\n";
22                 break;
23       default:  cout << "You can only choose the 100,";
24                 cout << "200, or 300.\n";
25    }
26    return 0;
27 }
```

Continued…

`break` and **`default`** statements in Pgm 4-25

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 100 [Enter]
That model has the following features:
    Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 200 [Enter]
That model has the following features:
    Stereo sound.
    Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 300 [Enter]
That model has the following features:
    Picture-in-a-picture.
    Stereo sound.
    Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 500 [Enter]
That model has the following features:
You can only choose the 100, 200, or 300.
```