SU
SINAI UNIVERSITY

CSW 232
Computer Programming (1)

SPRING 2024

Lecture 03 – Decision Making
Instructor: Dr. Tarek Abdul Hamid

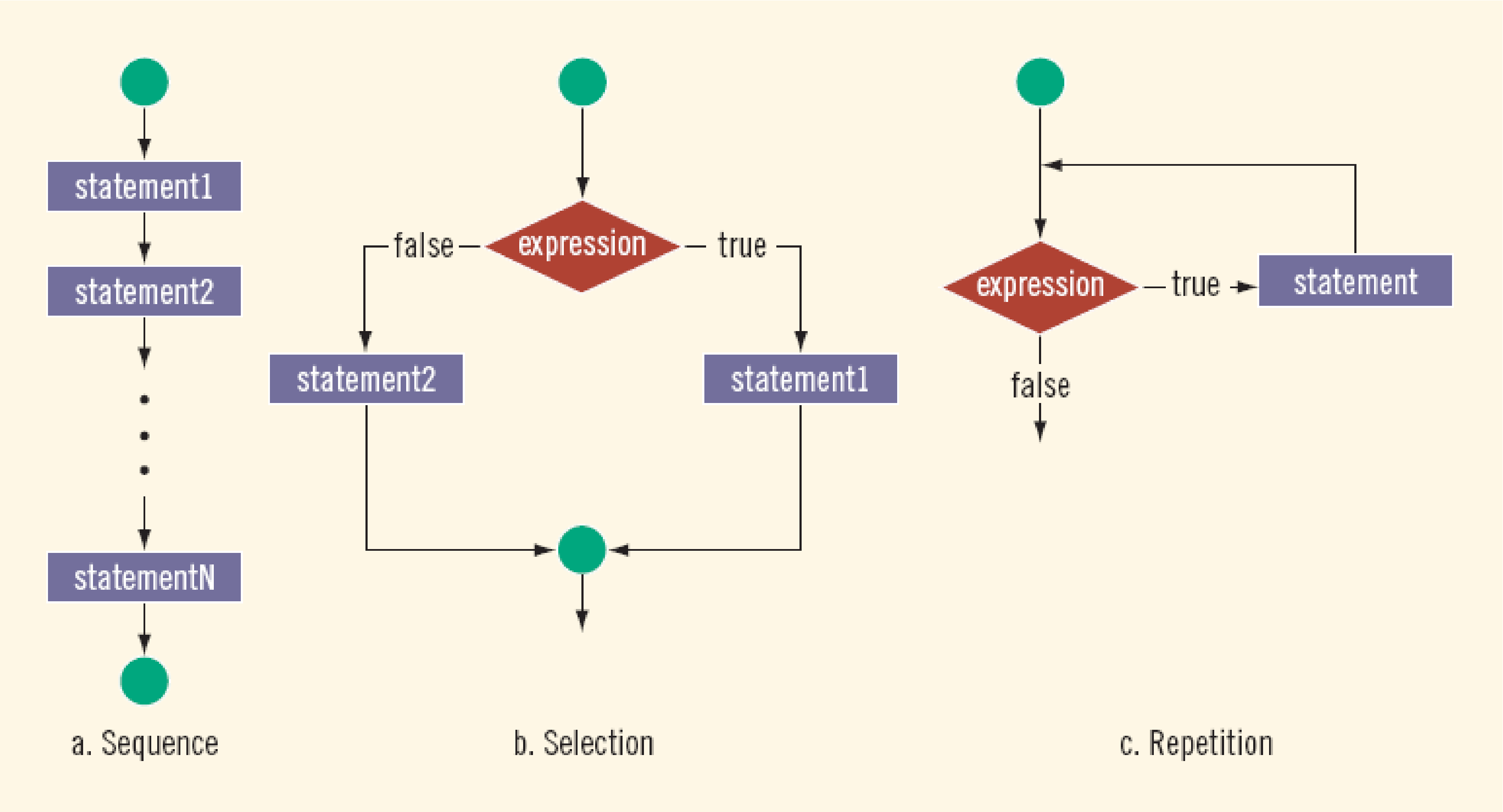
Control Structures

- A computer can proceed:
  - In sequence
  - **Selectively** (branch) - making a choice
  - **Repetitively** (iteratively) - looping
- Some statements are executed only if certain conditions are met
- A condition is met if it evaluates to `true`

Control Structures

Selection

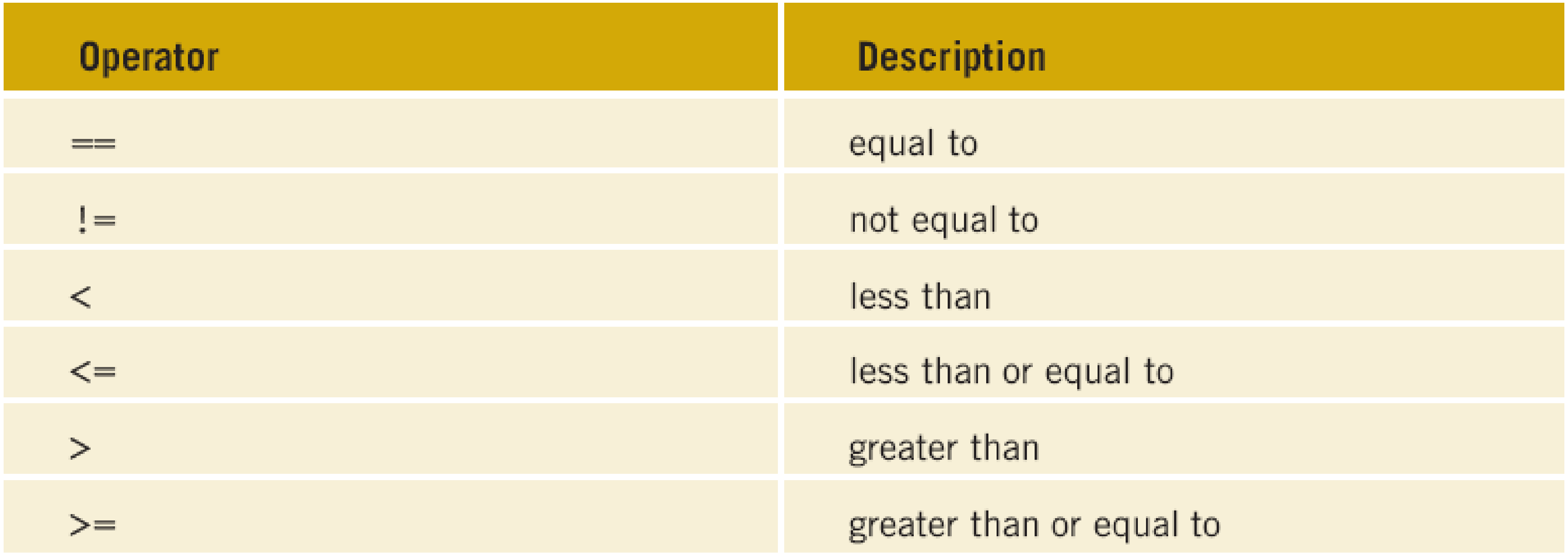
Control Structures



Relational Operators

- A condition is represented by a logical (Boolean) expression that can be `true` or `false`
- Relational operators:
  - Allow comparisons
  - Require two operands (binary)
  - Evaluate to `true` or `false`

Relational Operators

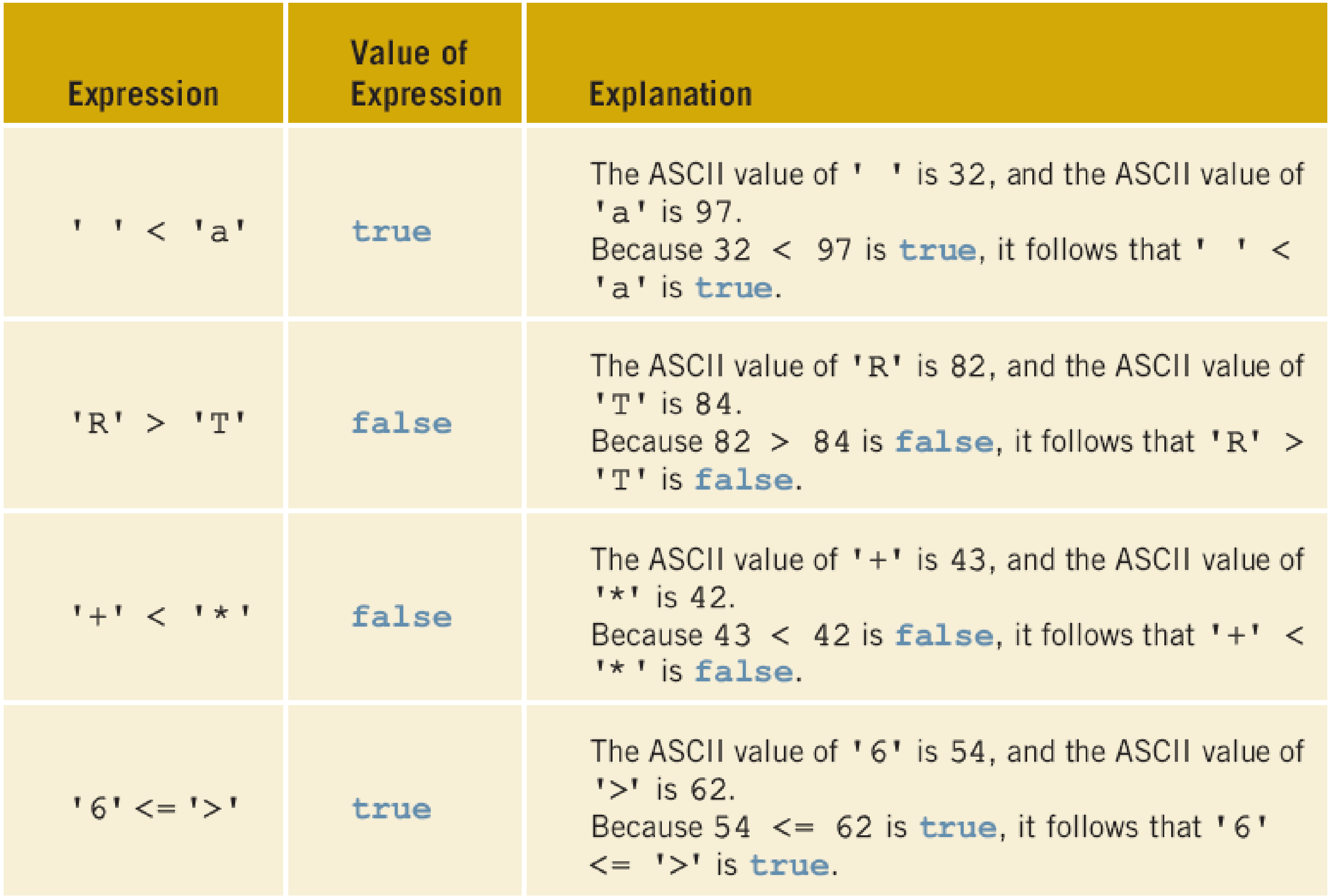


Operator	Description
==	equal to
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

Relational Operators and Simple Data Types

- You can use the relational operators with all three simple data types:
  - `8 < 15` evaluates to `true`
  - `6 != 6` evaluates to `false`
  - `2.5 > 5.8` evaluates to `false`
  - `5.9 <= 7.5` evaluates to `true`

Comparing Characters



Expression	Value of Expression	Explanation
' ' < 'a'	true	The ASCII value of ' ' is 32, and the ASCII value of 'a' is 97. Because 32 < 97 is true, it follows that ' ' < 'a' is true.
'R' > 'T'	false	The ASCII value of 'R' is 82, and the ASCII value of 'T' is 84. Because 82 > 84 is false, it follows that 'R' > 'T' is false.
'+' < '*'	false	The ASCII value of '+' is 43, and the ASCII value of '*' is 42. Because 43 < 42 is false, it follows that '+' < '*' is false.
'6' <= '>'	true	The ASCII value of '6' is 54, and the ASCII value of '>' is 62. Because 54 <= 62 is true, it follows that '6' <= '>' is true.

Relational Operators and the `string` Type

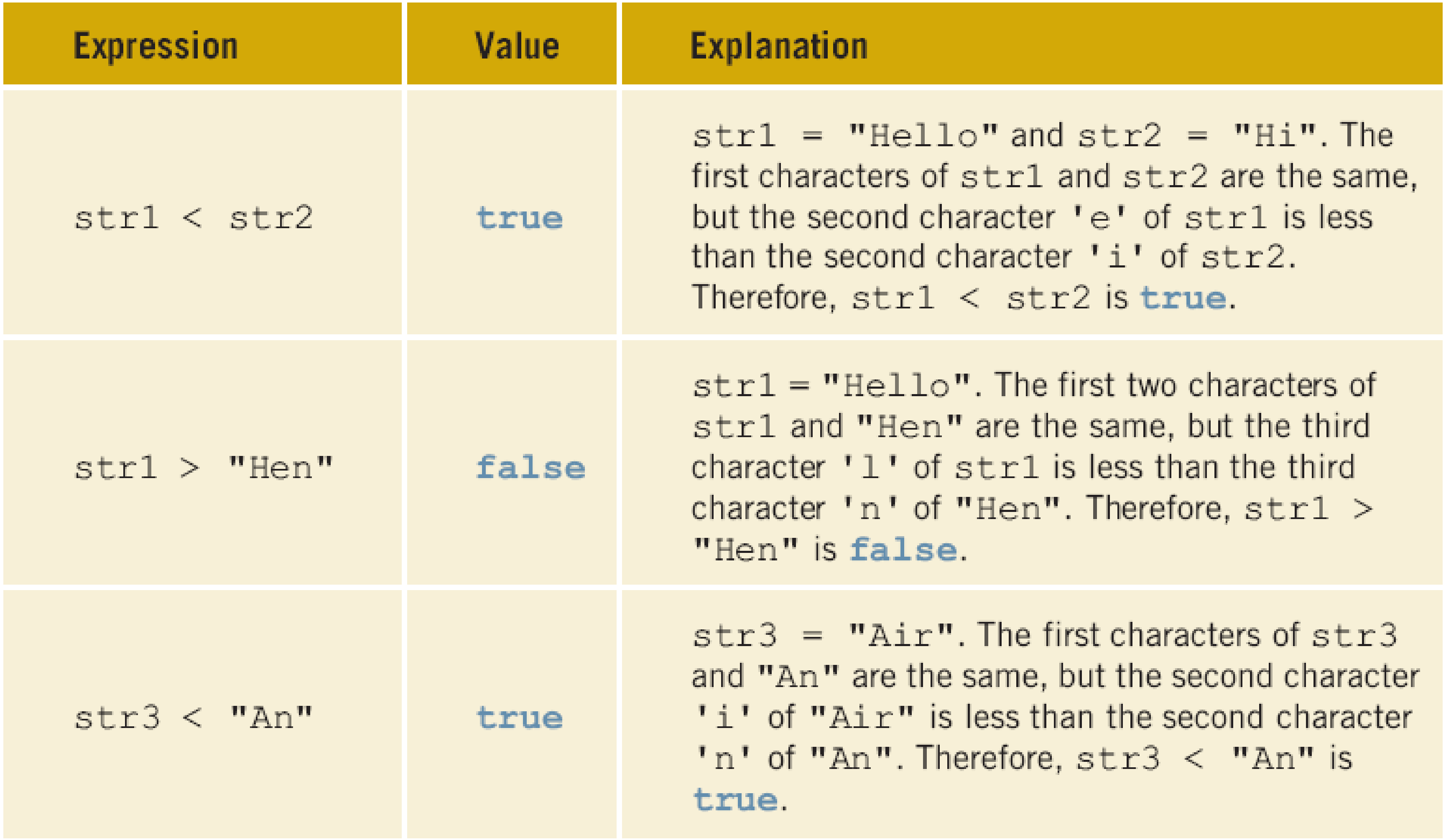
- Relational operators can be applied to strings
- Strings are compared character by character, starting with the first character
- Comparison continues until either a mismatch is found or all characters are found equal
- If two strings of different lengths are compared and the comparison is equal to the last character of the shorter string
  - The shorter string is less than the larger string

Relational Operators and the `string` Type

- Suppose we have the following declarations:

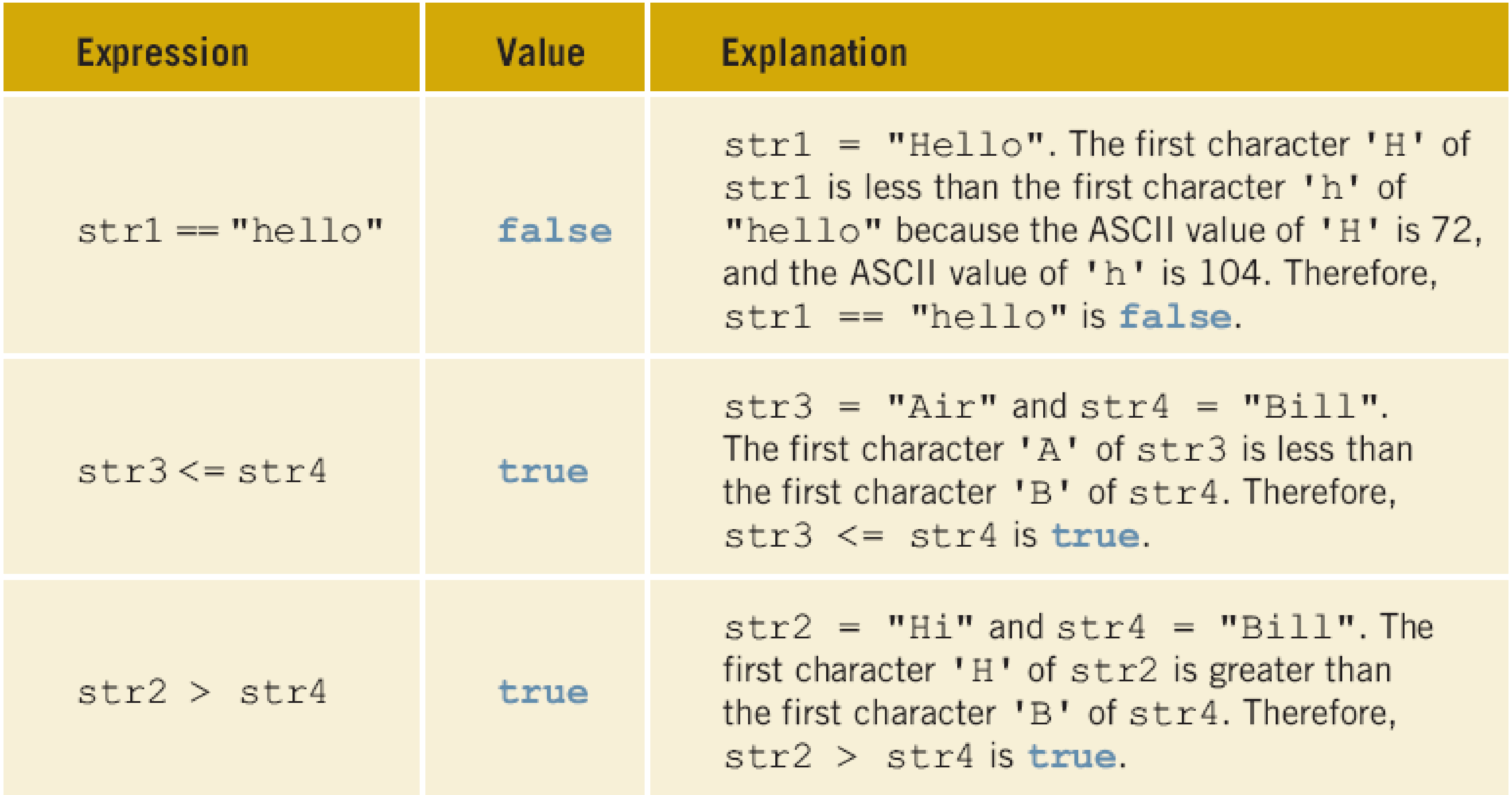
```
string str1 = "Hello";
string str2 = "Hi";
string str3 = "Air";
string str4 = "Bill";
string str4 = "Big";
```

Relational Operators and the string Type



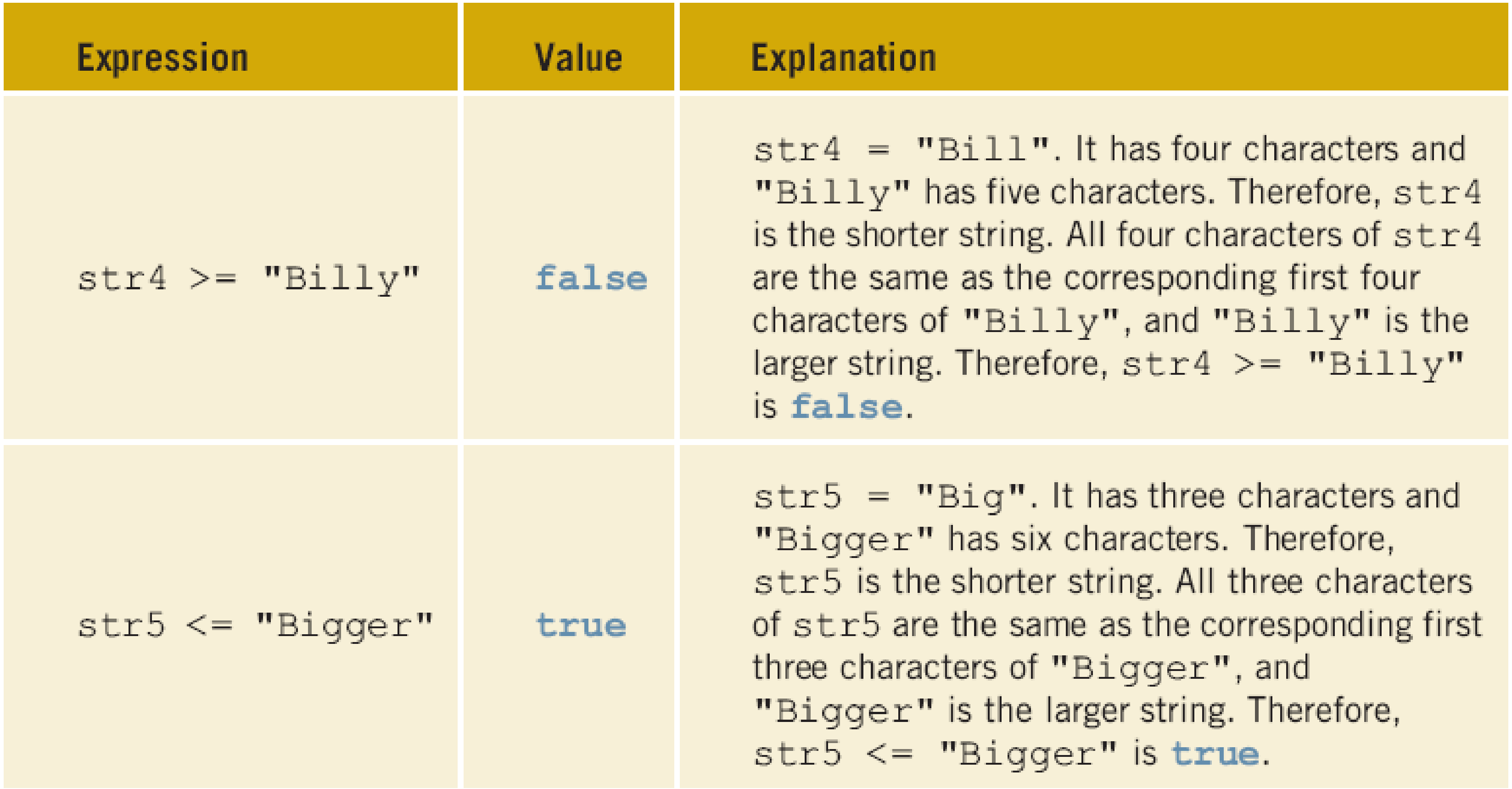
Expression	Value	Explanation
str1 < str2	true	str1 = "Hello" and str2 = "Hi". The first characters of str1 and str2 are the same, but the second character 'e' of str1 is less than the second character 'i' of str2. Therefore, str1 < str2 is true.
str1 > "Hen"	false	str1 = "Hello". The first two characters of str1 and "Hen" are the same, but the third character 'l' of str1 is less than the third character 'n' of "Hen". Therefore, str1 > "Hen" is false.
str3 < "An"	true	str3 = "Air". The first characters of str3 and "An" are the same, but the second character 'i' of "Air" is less than the second character 'n' of "An". Therefore, str3 < "An" is true.

Relational Operators and the `string` Type



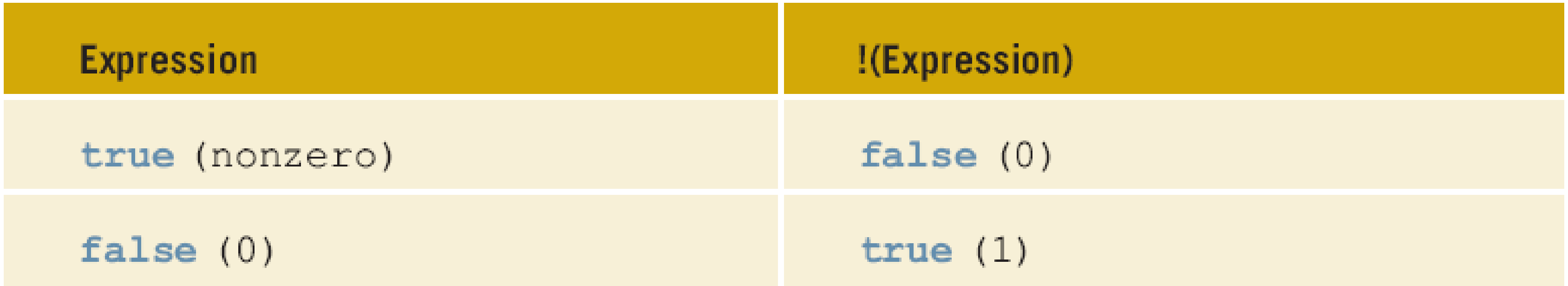
Expression	Value	Explanation
`str1 == "hello"`	`false`	`str1 = "Hello"`. The first character `'H'` of `str1` is less than the first character `'h'` of `"hello"` because the ASCII value of `'H'` is 72, and the ASCII value of `'h'` is 104. Therefore, `str1 == "hello"` is `false`.
`str3 <= str4`	`true`	`str3 = "Air"` and `str4 = "Bill"`. The first character `'A'` of `str3` is less than the first character `'B'` of `str4`. Therefore, `str3 <= str4` is `true`.
`str2 > str4`	`true`	`str2 = "Hi"` and `str4 = "Bill"`. The first character `'H'` of `str2` is greater than the first character `'B'` of `str4`. Therefore, `str2 > str4` is `true`.

Relational Operators and the `string` Type

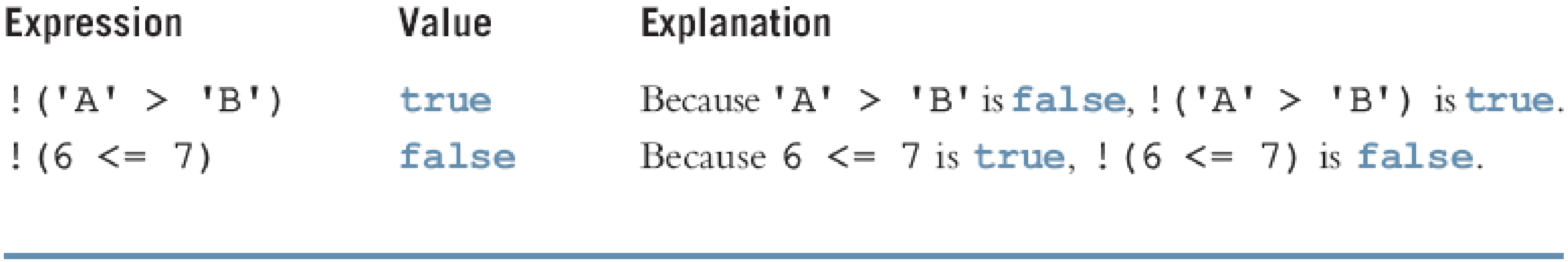


Expression	Value	Explanation
`str4 >= "Billy"`	`false`	`str4 = "Bill"`. It has four characters and `"Billy"` has five characters. Therefore, `str4` is the shorter string. All four characters of `str4` are the same as the corresponding first four characters of `"Billy"`, and `"Billy"` is the larger string. Therefore, `str4 >= "Billy"` is `false`.
`str5 <= "Bigger"`	`true`	`str5 = "Big"`. It has three characters and `"Bigger"` has six characters. Therefore, `str5` is the shorter string. All three characters of `str5` are the same as the corresponding first three characters of `"Bigger"`, and `"Bigger"` is the larger string. Therefore, `str5 <= "Bigger"` is `true`.

Logical (Boolean) Operators and Logical Expressions



Expression	!(Expression)
true (nonzero)	false (0)
false (0)	true (1)



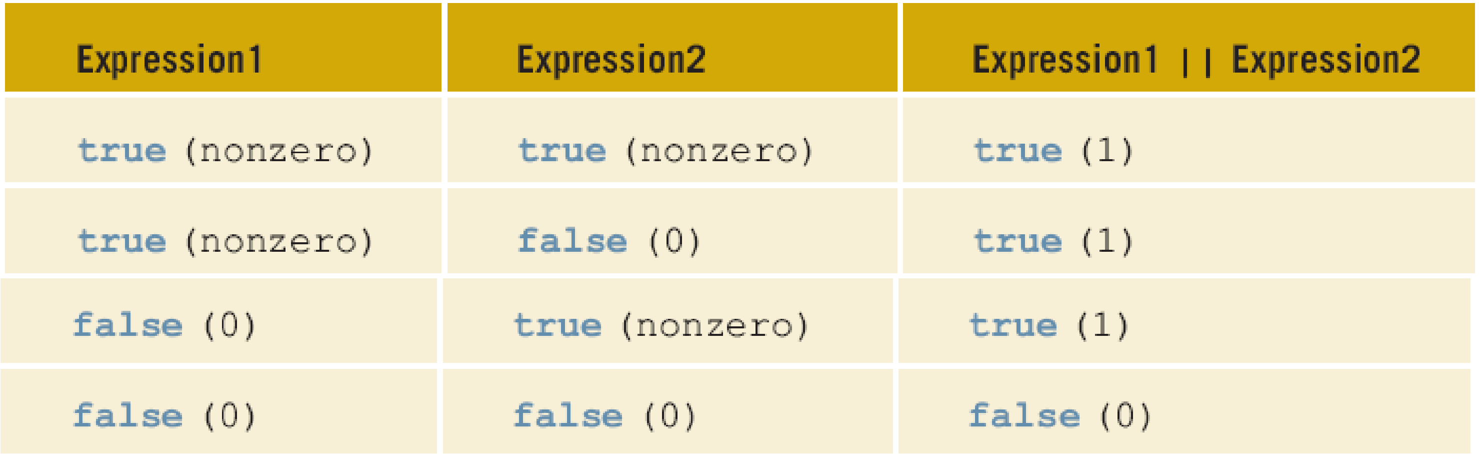
Expression	Value	Explanation
!('A' > 'B')	true	Because 'A' > 'B' is false, !('A' > 'B') is true.
!(6 <= 7)	false	Because 6 <= 7 is true, !(6 <= 7) is false.

Logical (Boolean) Operators and Logical Expressions

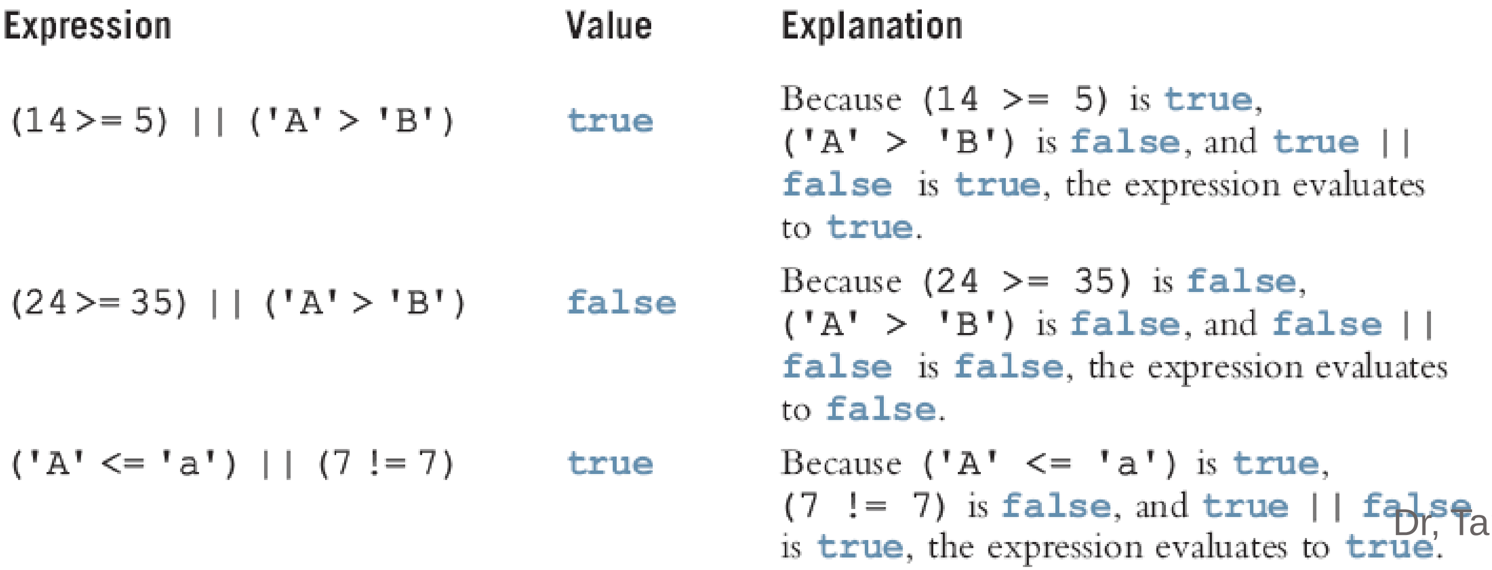
Expression1	Expression2	Expression1 && Expression2
true (nonzero)	true (nonzero)	true (1)
true (nonzero)	false (0)	false (0)
false (0)	true (nonzero)	false (0)
false (0)	false (0)	false (0)

Expression	Value	Explanation
(14 >= 5) && ('A' < 'B')	true	Because (14 >= 5) is true, ('A' < 'B') is true, and true && true is true, the expression evaluates to true.
(24 >= 35) && ('A' < 'B')	false	Because (24 >= 35) is false, ('A' < 'B') is true, and false && true is false, the expression evaluates to false.

Logical (Boolean) Operators and Logical Expressions



Expression1	Expression2	Expression1 || Expression2
true (nonzero)	true (nonzero)	true (1)
true (nonzero)	false (0)	true (1)
false (0)	true (nonzero)	true (1)
false (0)	false (0)	false (0)

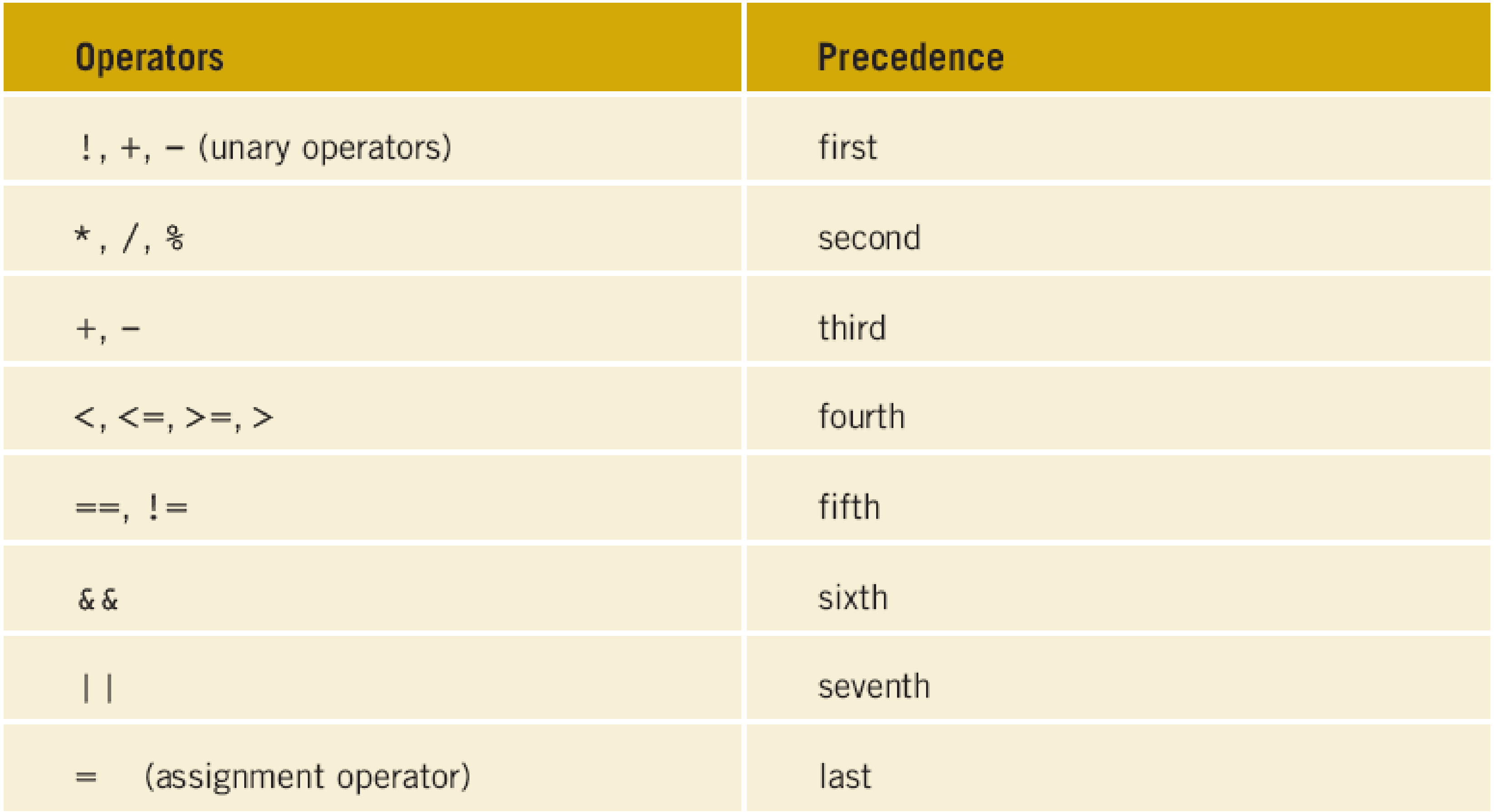


Expression	Value	Explanation
(14 >= 5) || ('A' > 'B')	true	Because (14 >= 5) is true, ('A' > 'B') is false, and true || false is true, the expression evaluates to true.
(24 >= 35) || ('A' > 'B')	false	Because (24 >= 35) is false, ('A' > 'B') is false, and false || false is false, the expression evaluates to false.
('A' <= 'a') || (7 != 7)	true	Because ('A' <= 'a') is true, (7 != 7) is false, and true || false is true, the expression evaluates to true.

Order of Precedence

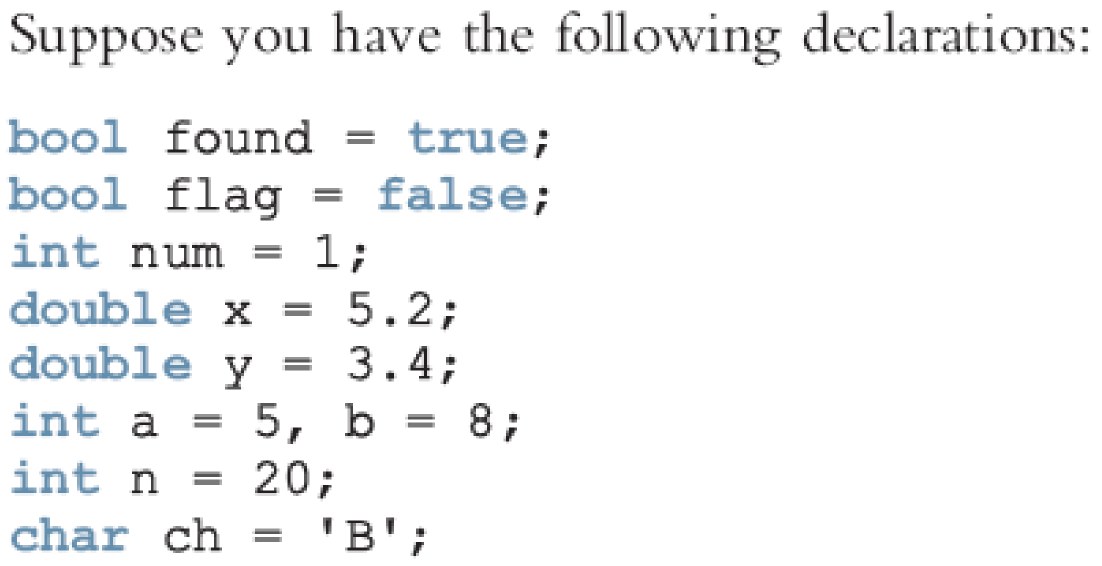
- Relational and logical operators are evaluated from left to right
- The associativity is left to right
- Parentheses can override precedence

Order of Precedence



Operators	Precedence
!, +, - (unary operators)	first
*, /, %	second
+, -	third
<, <=, >=, >	fourth
==, !=	fifth
&&	sixth
||	seventh
= (assignment operator)	last

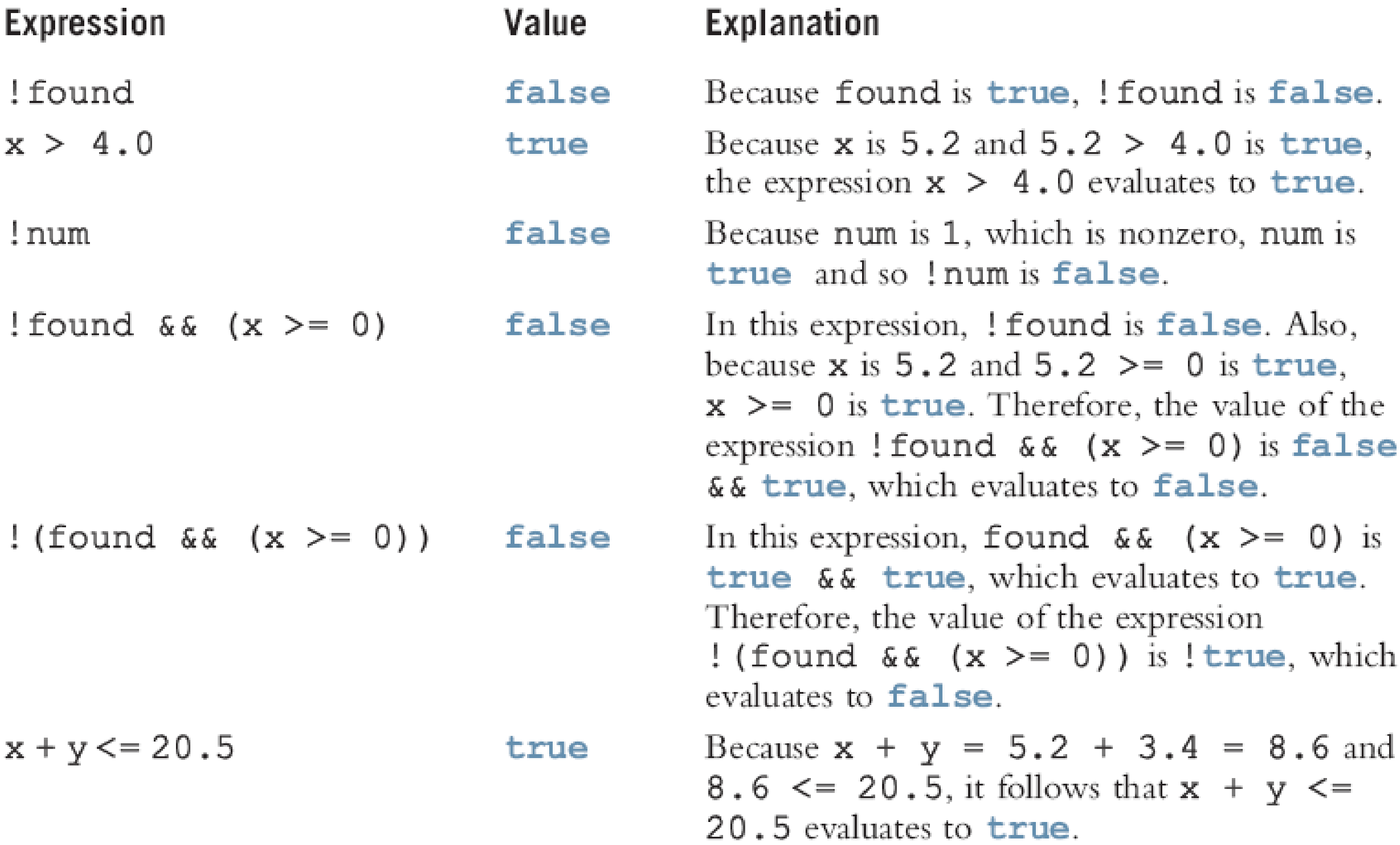
Order of Precedence



Suppose you have the following declarations:

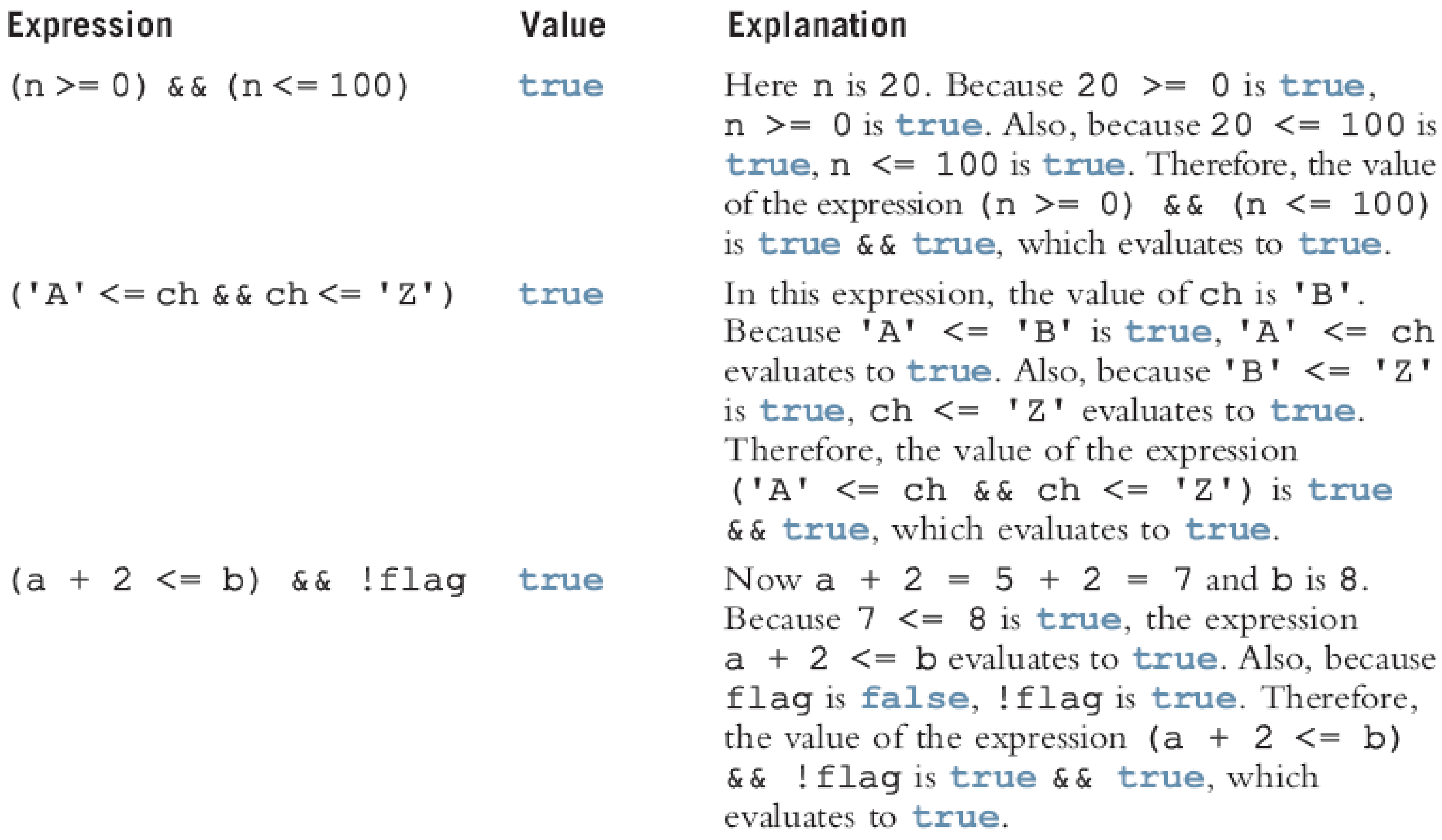
```
bool found = true;
bool flag = false;
int num = 1;
double x = 5.2;
double y = 3.4;
int a = 5, b = 8;
int n = 20;
char ch = 'B';
```

Order of Precedence



Expression	Value	Explanation
!found	false	Because found is true, !found is false.
x > 4.0	true	Because x is 5.2 and 5.2 > 4.0 is true, the expression x > 4.0 evaluates to true.
!num	false	Because num is 1, which is nonzero, num is true and so !num is false.
!found && (x >= 0)	false	In this expression, !found is false. Also, because x is 5.2 and 5.2 >= 0 is true, x >= 0 is true. Therefore, the value of the expression !found && (x >= 0) is false && true, which evaluates to false.
!(found && (x >= 0))	false	In this expression, found && (x >= 0) is true && true, which evaluates to true. Therefore, the value of the expression !(found && (x >= 0)) is !true, which evaluates to false.
x + y <= 20.5	true	Because x + y = 5.2 + 3.4 = 8.6 and 8.6 <= 20.5, it follows that x + y <= 20.5 evaluates to true.

Order of Precedence



Expression	Value	Explanation
(n >= 0) && (n <= 100)	true	Here n is 20. Because 20 >= 0 is true, n >= 0 is true. Also, because 20 <= 100 is true, n <= 100 is true. Therefore, the value of the expression (n >= 0) && (n <= 100) is true && true, which evaluates to true.
('A' <= ch && ch <= 'Z')	true	In this expression, the value of ch is 'B'. Because 'A' <= 'B' is true, 'A' <= ch evaluates to true. Also, because 'B' <= 'Z' is true, ch <= 'Z' evaluates to true. Therefore, the value of the expression ('A' <= ch && ch <= 'Z') is true && true, which evaluates to true.
(a + 2 <= b) && !flag	true	Now a + 2 = 5 + 2 = 7 and b is 8. Because 7 <= 8 is true, the expression a + 2 <= b evaluates to true. Also, because flag is false, !flag is true. Therefore, the value of the expression (a + 2 <= b) && !flag is true && true, which evaluates to true.

Short-Circuit Evaluation

- Short-circuit evaluation: evaluation of a logical expression stops as soon as the value of the expression is known
- Example:

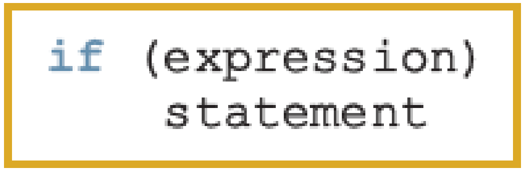
```
(age >= 21) || ( x == 5)       //Line 1
(grade == 'A') && (x >= 7)     //Line 2
```

Selection: `if` and `if...else`

- One-Way Selection
- Two-Way Selection
- Compound (Block of) Statements
- Multiple Selections: Nested `if`
- Comparing `if...else` Statements with a Series of `if` Statements

One-Way Selection

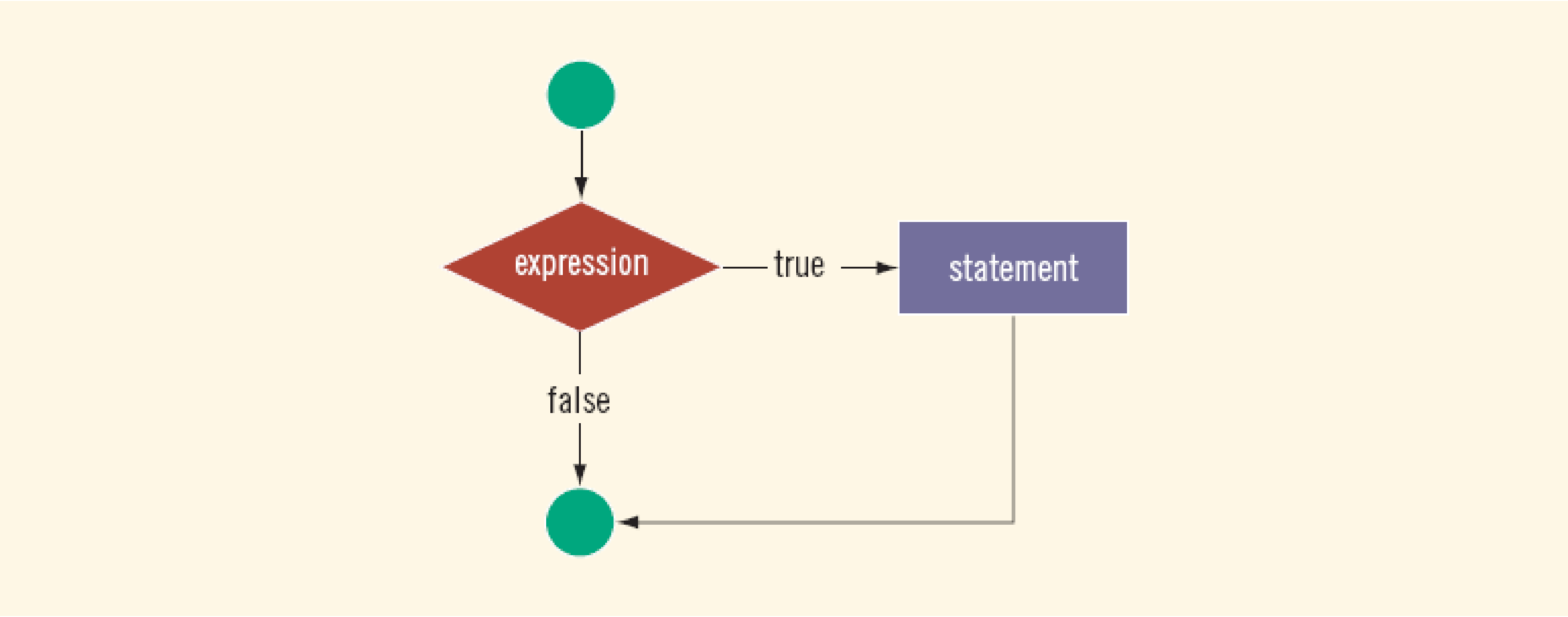
- The syntax of one-way selection is:



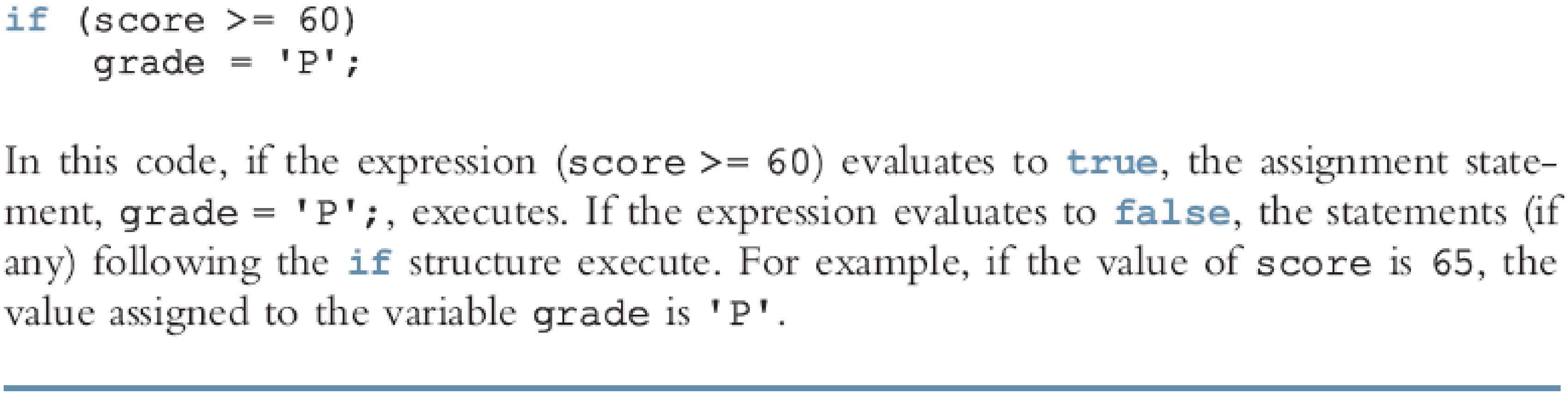
```
if (expression)
    statement
```

- The statement is executed if the value of the expression is `true`
- The statement is bypassed if the value is `false`; program goes to the next statement
- `if` is a reserved word

One-Way Selection



One-Way Selection



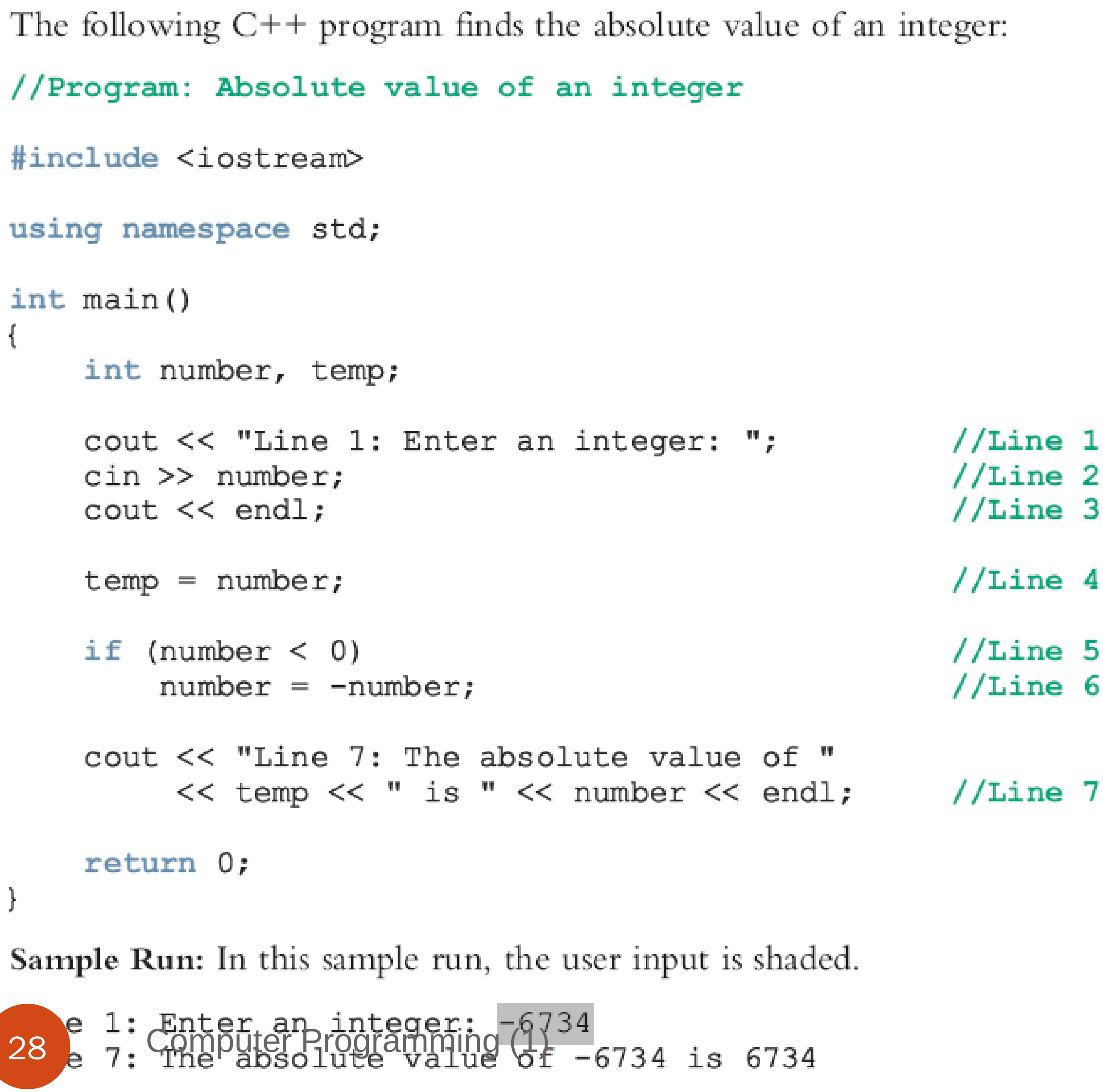
```
if (score >= 60)
    grade = 'P';
```

In this code, if the expression (`score >= 60`) evaluates to **`true`**, the assignment statement, `grade = 'P';`, executes. If the expression evaluates to **`false`**, the statements (if any) following the **`if`** structure execute. For example, if the value of `score` is `65`, the value assigned to the variable `grade` is `'P'`.

Example

Write a program to input an integer and print its absolute value

Example



The following C++ program finds the absolute value of an integer:

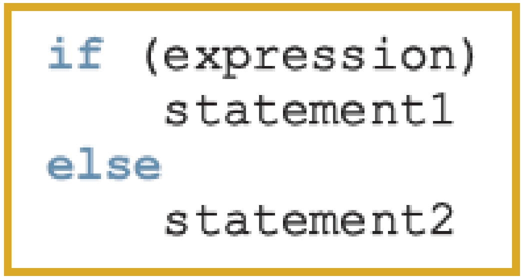
```
//Program: Absolute value of an integer

#include <iostream>

using namespace std;

int main()
{
    int number, temp;

    cout << "Line 1: Enter an integer: ";        //Line 1
    cin >> number;                                //Line 2
    cout << endl;                                 //Line 3

    temp = number;                                //Line 4

    if (number < 0)                               //Line 5
        number = -number;                         //Line 6

    cout << "Line 7: The absolute value of "
         << temp << " is " << number << endl;     //Line 7

    return 0;
}
```

Sample Run: In this sample run, the user input is shaded.

```
e 1: Enter an integer: -6734
e 7: The absolute value of -6734 is 6734
```

Two-Way Selection

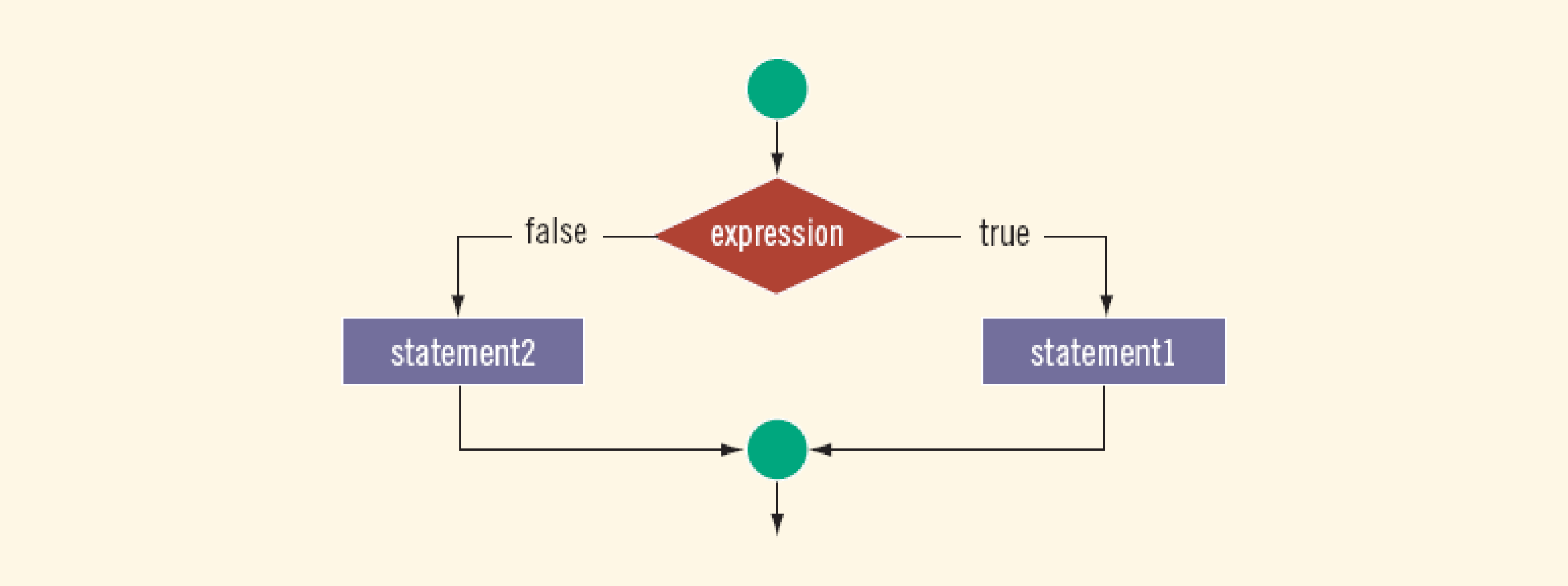
- Two-way selection takes the form:



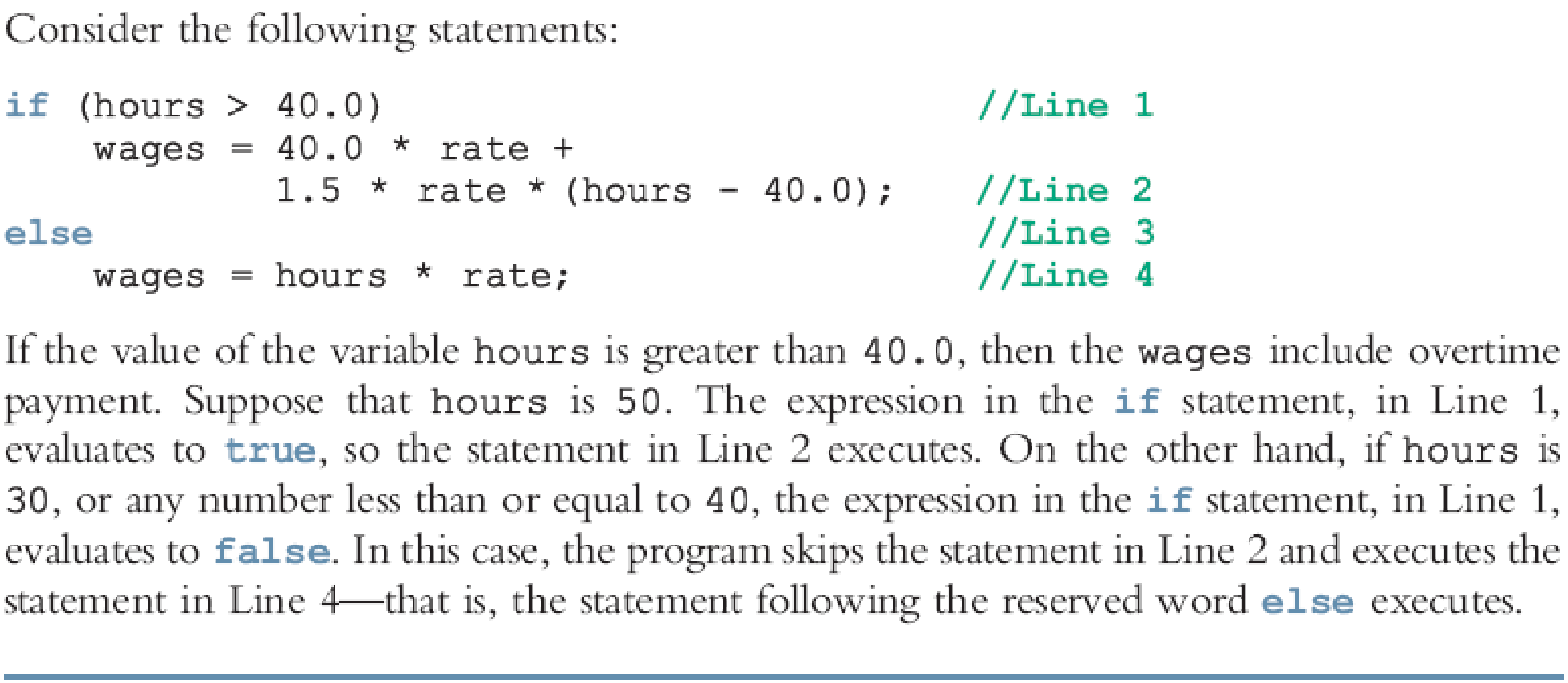
```
if (expression)
     statement1
else
     statement2
```

- If expression is `true`, `statement1` is executed; otherwise, `statement2` is executed
    - `statement1` and `statement2` are any C++ statements
- `else` is a reserved word

Two-Way Selection



Two-Way Selection

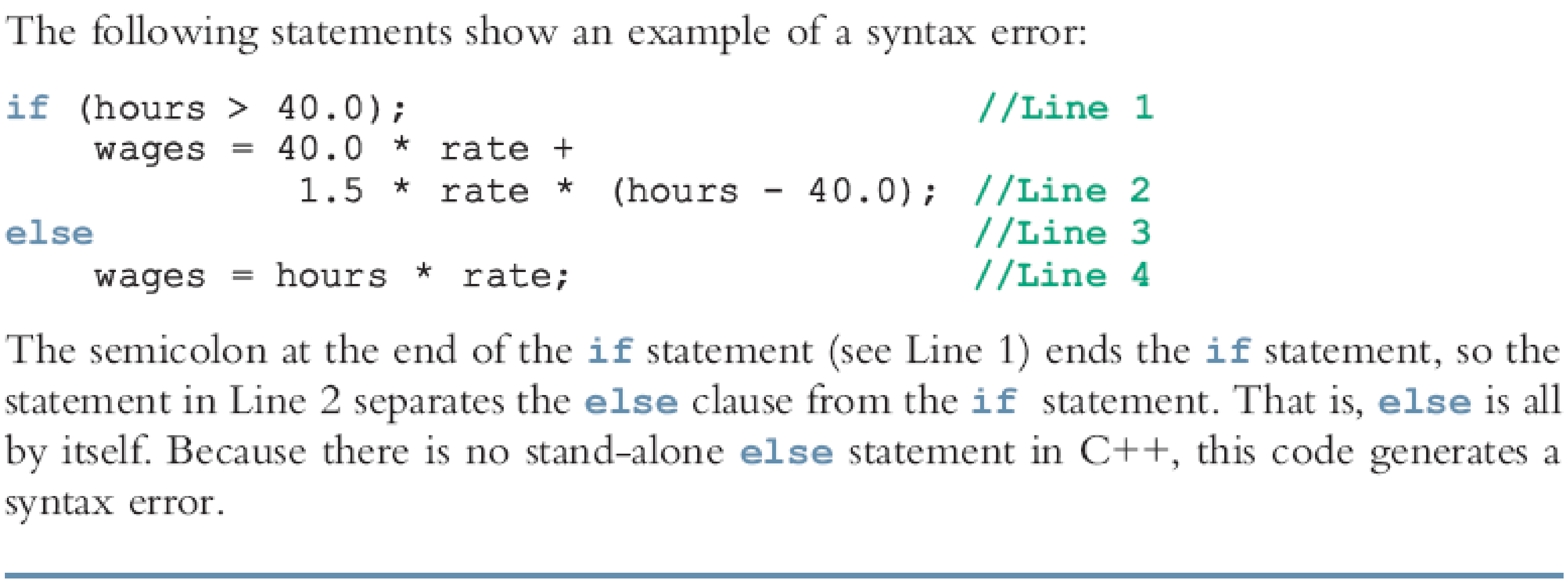


Consider the following statements:

```
if (hours > 40.0)                                 //Line 1
    wages = 40.0 * rate +
            1.5 * rate * (hours - 40.0);          //Line 2
else                                              //Line 3
    wages = hours * rate;                         //Line 4
```

If the value of the variable `hours` is greater than `40.0`, then the `wages` include overtime payment. Suppose that `hours` is `50`. The expression in the `if` statement, in Line 1, evaluates to `true`, so the statement in Line 2 executes. On the other hand, if `hours` is `30`, or any number less than or equal to `40`, the expression in the `if` statement, in Line 1, evaluates to `false`. In this case, the program skips the statement in Line 2 and executes the statement in Line 4—that is, the statement following the reserved word `else` executes.

Two-Way Selection



The following statements show an example of a syntax error:

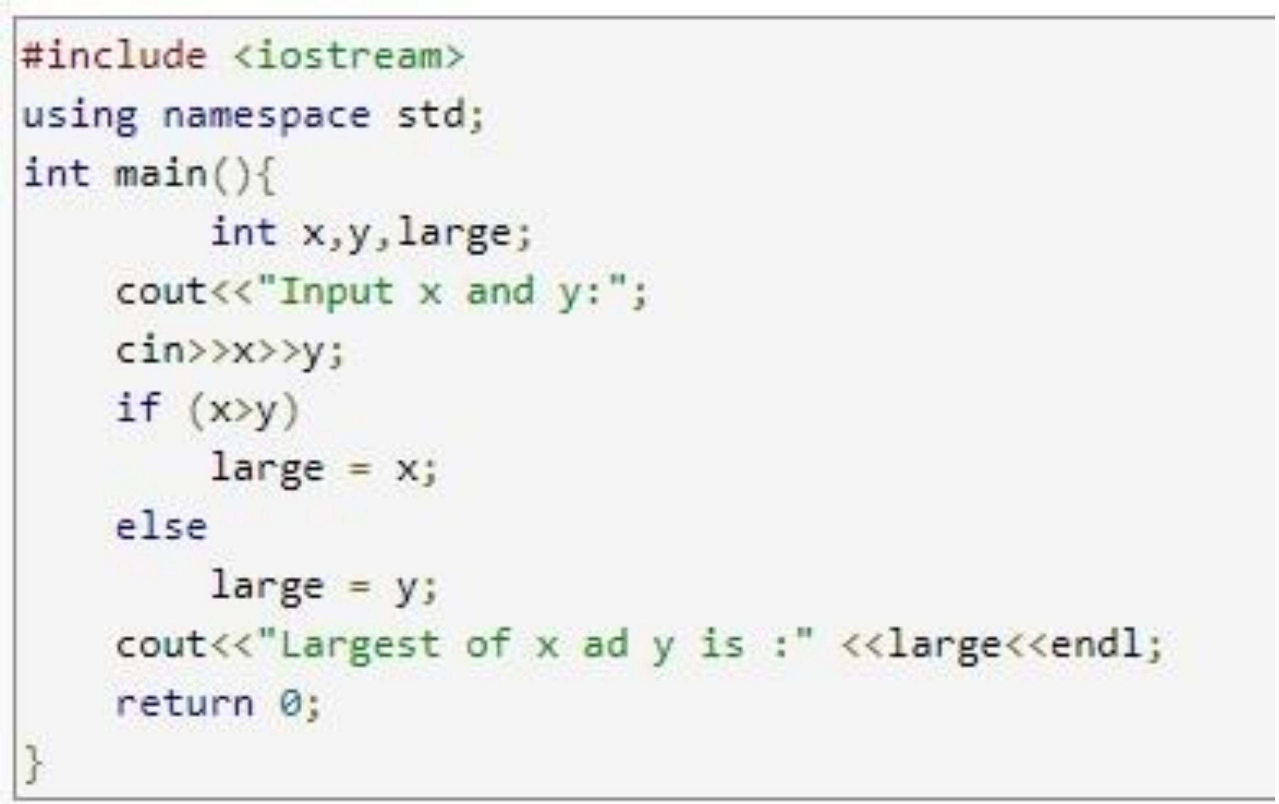
```
if (hours > 40.0);                          //Line 1
    wages = 40.0 * rate +
             1.5 * rate * (hours - 40.0);  //Line 2
else                                        //Line 3
    wages = hours * rate;                   //Line 4
```

The semicolon at the end of the if statement (see Line 1) ends the if statement, so the statement in Line 2 separates the else clause from the if statement. That is, else is all by itself. Because there is no stand-alone else statement in C++, this code generates a syntax error.

Example

Write a program to input two integers and print its largest value

Example

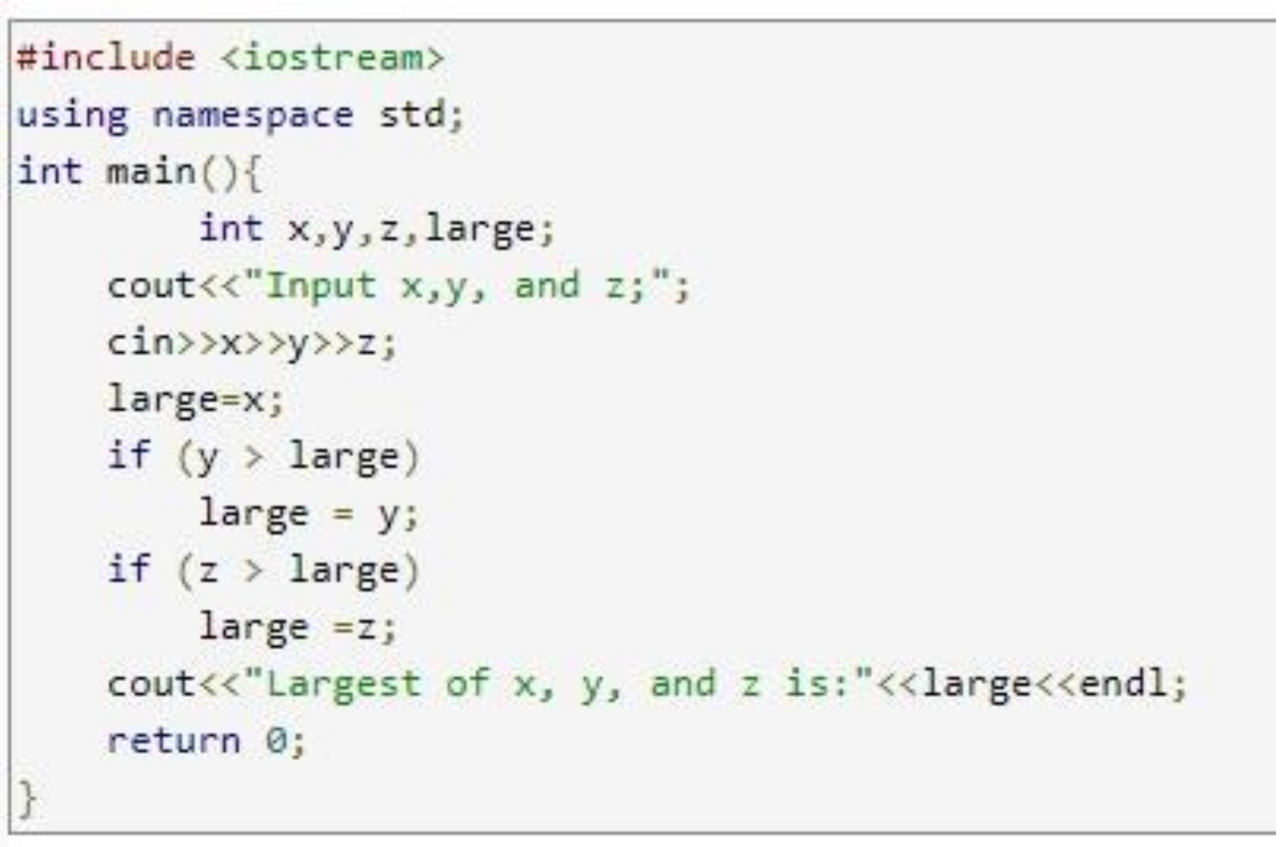


```
#include <iostream>
using namespace std;
int main(){
        int x,y,large;
    cout<<"Input x and y:";
    cin>>x>>y;
    if (x>y)
        large = x;
    else
        large = y;
    cout<<"Largest of x ad y is :" <<large<<endl;
    return 0;
}
```

Example

Write a program to input Three integers and print its largest value

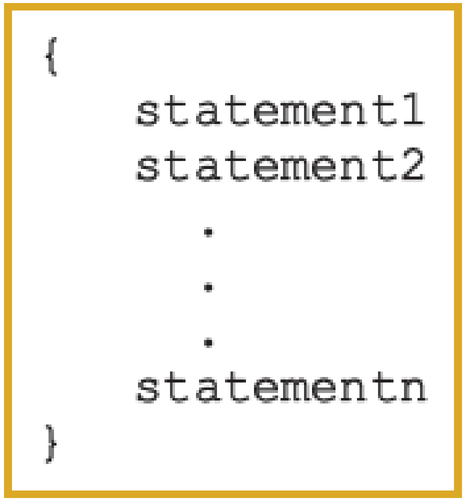
Example



```
#include <iostream>
using namespace std;
int main(){
        int x,y,z,large;
    cout<<"Input x,y, and z;";
    cin>>x>>y>>z;
    large=x;
    if (y > large)
        large = y;
    if (z > large)
        large =z;
    cout<<"Largest of x, y, and z is:"<<large<<endl;
    return 0;
}
```

Compound (Block of) Statement

- Compound statement (block of statements):



```
{
     statement1
     statement2
         .
         .
         .
     statementn
}
```

- A compound statement is a single statement

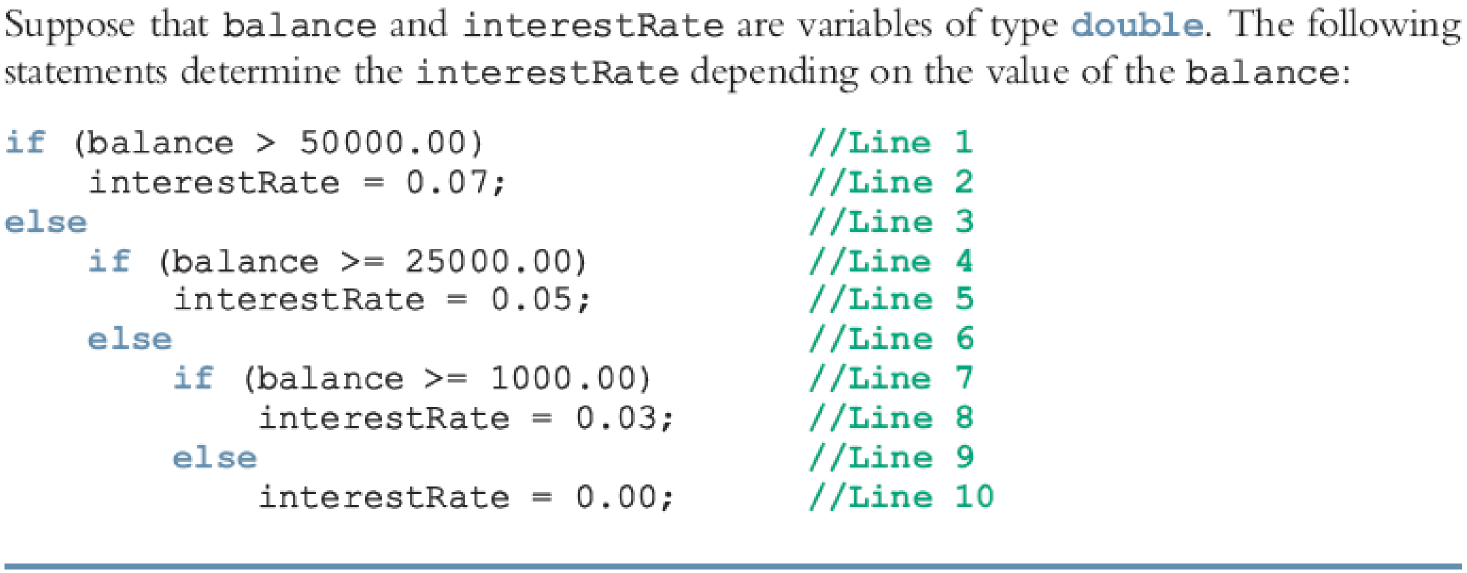
Compound (Block of) Statement

```
if (age > 18)
{
  cout << "Eligible to vote." << endl;
  cout << "No longer a minor." << endl;
}
else
{
  cout << "Not eligible to vote." << endl;
  cout << "Still a minor." << endl;
}
```

Multiple Selections: Nested if

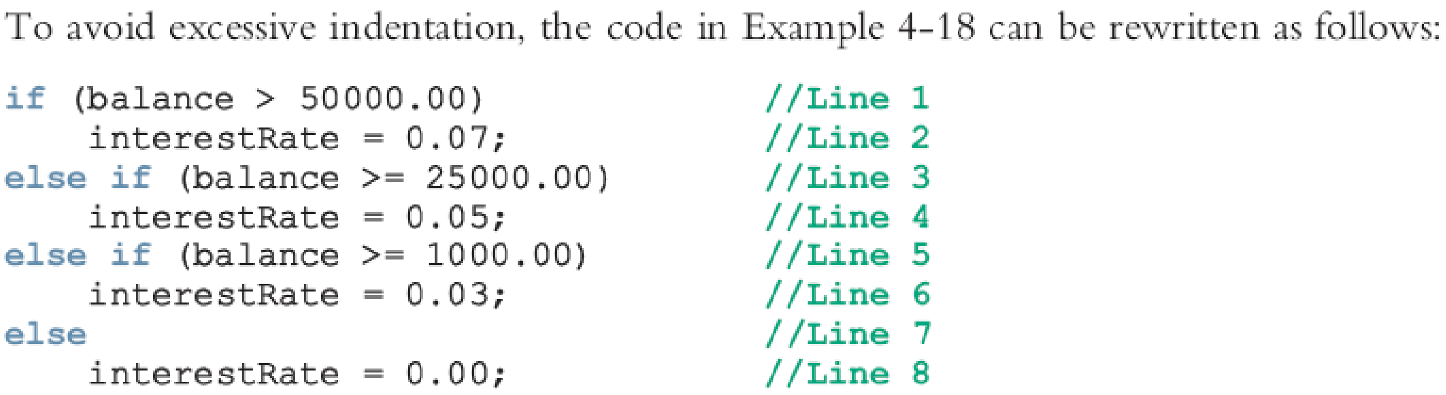
- Nesting: one control statement in another
- An `else` is associated with the most recent `if` that has not been paired with an `else`

Example



Suppose that balance and interestRate are variables of type double. The following statements determine the interestRate depending on the value of the balance:

```
if (balance > 50000.00)                 //Line 1
    interestRate = 0.07;                //Line 2
else                                    //Line 3
    if (balance >= 25000.00)            //Line 4
        interestRate = 0.05;            //Line 5
    else                                //Line 6
        if (balance >= 1000.00)         //Line 7
            interestRate = 0.03;        //Line 8
        else                            //Line 9
            interestRate = 0.00;        //Line 10
```



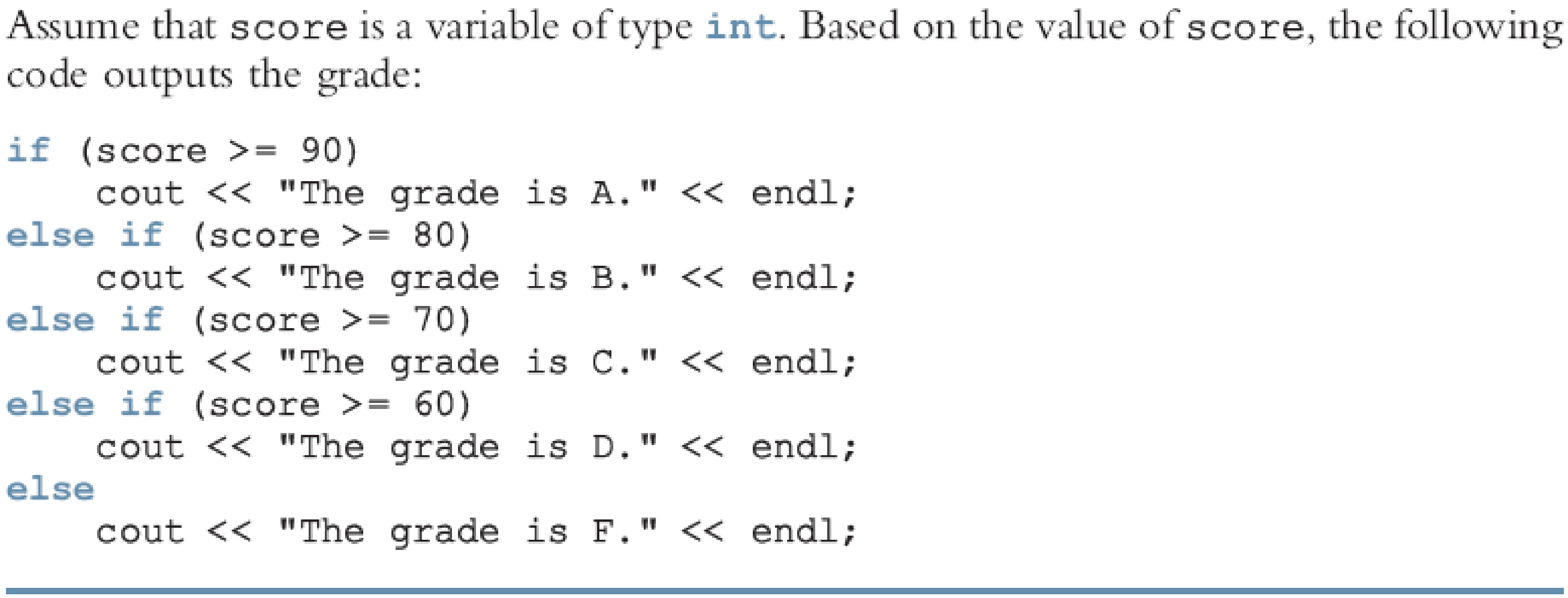
To avoid excessive indentation, the code in Example 4-18 can be rewritten as follows:

```
if (balance > 50000.00)              //Line 1
    interestRate = 0.07;             //Line 2
else if (balance >= 25000.00)        //Line 3
    interestRate = 0.05;             //Line 4
else if (balance >= 1000.00)         //Line 5
    interestRate = 0.03;             //Line 6
else                                 //Line 7
    interestRate = 0.00;             //Line 8
```

Example

Write a program to input the score in numbers and print the grade

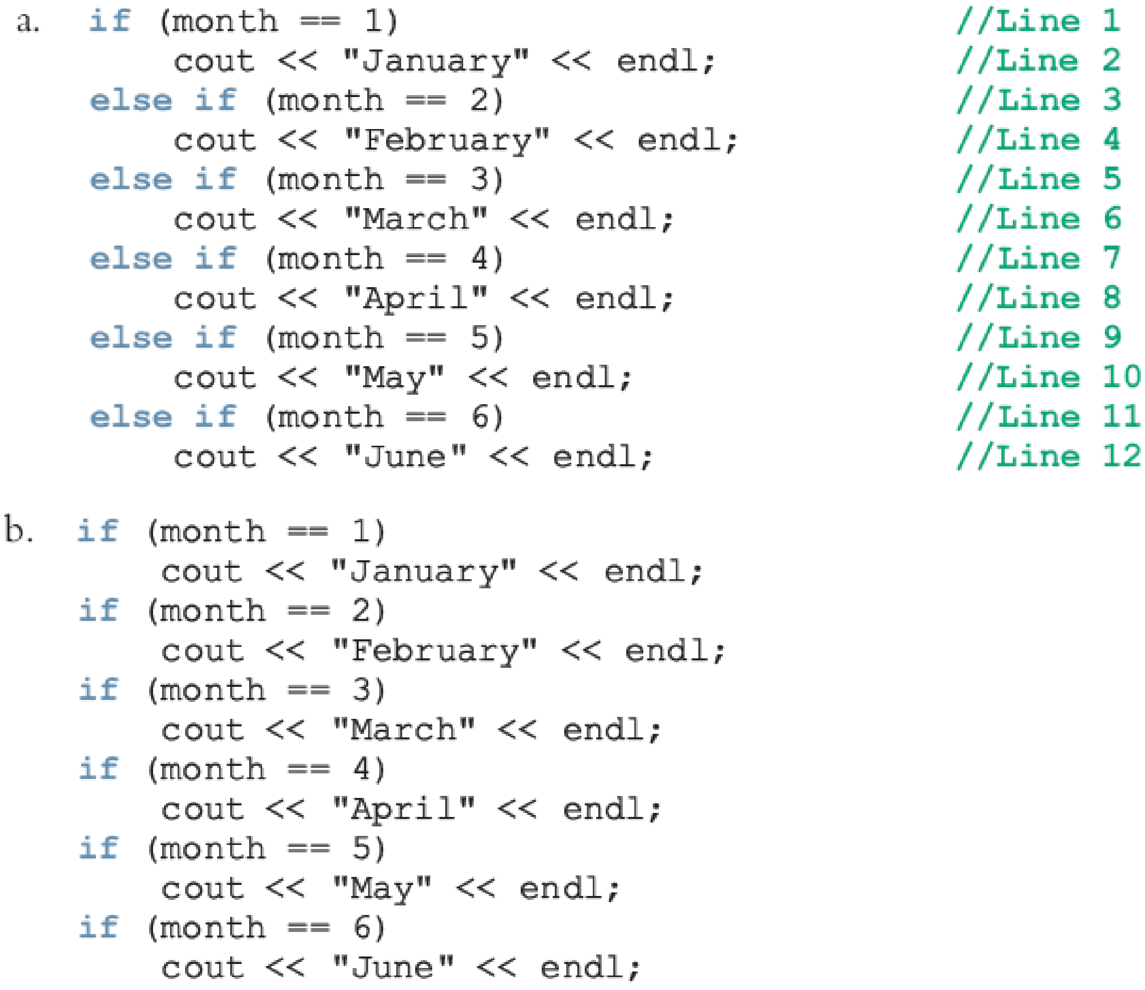
Example



Assume that score is a variable of type int. Based on the value of score, the following code outputs the grade:

```
if (score >= 90)
    cout << "The grade is A." << endl;
else if (score >= 80)
    cout << "The grade is B." << endl;
else if (score >= 70)
    cout << "The grade is C." << endl;
else if (score >= 60)
    cout << "The grade is D." << endl;
else
    cout << "The grade is F." << endl;
```

Comparing if…else Statements with a Series of if Statements



```
a.  if (month == 1)                          //Line 1
        cout << "January" << endl;           //Line 2
    else if (month == 2)                     //Line 3
        cout << "February" << endl;          //Line 4
    else if (month == 3)                     //Line 5
        cout << "March" << endl;             //Line 6
    else if (month == 4)                     //Line 7
        cout << "April" << endl;             //Line 8
    else if (month == 5)                     //Line 9
        cout << "May" << endl;               //Line 10
    else if (month == 6)                     //Line 11
        cout << "June" << endl;              //Line 12

b.  if (month == 1)
        cout << "January" << endl;
    if (month == 2)
        cout << "February" << endl;
    if (month == 3)
        cout << "March" << endl;
    if (month == 4)
        cout << "April" << endl;
    if (month == 5)
        cout << "May" << endl;
    if (month == 6)
        cout << "June" << endl;
```

Confusion Between == and =

- C++ allows you to use any expression that can be evaluated to either `true` or `false` as an expression in the `if` statement:

  ```
  if (x = 5)
      cout << "The value is five." << endl;
  ```

- The appearance of = in place of == resembles a *silent killer*
  - It is not a syntax error
  - It is a logical error

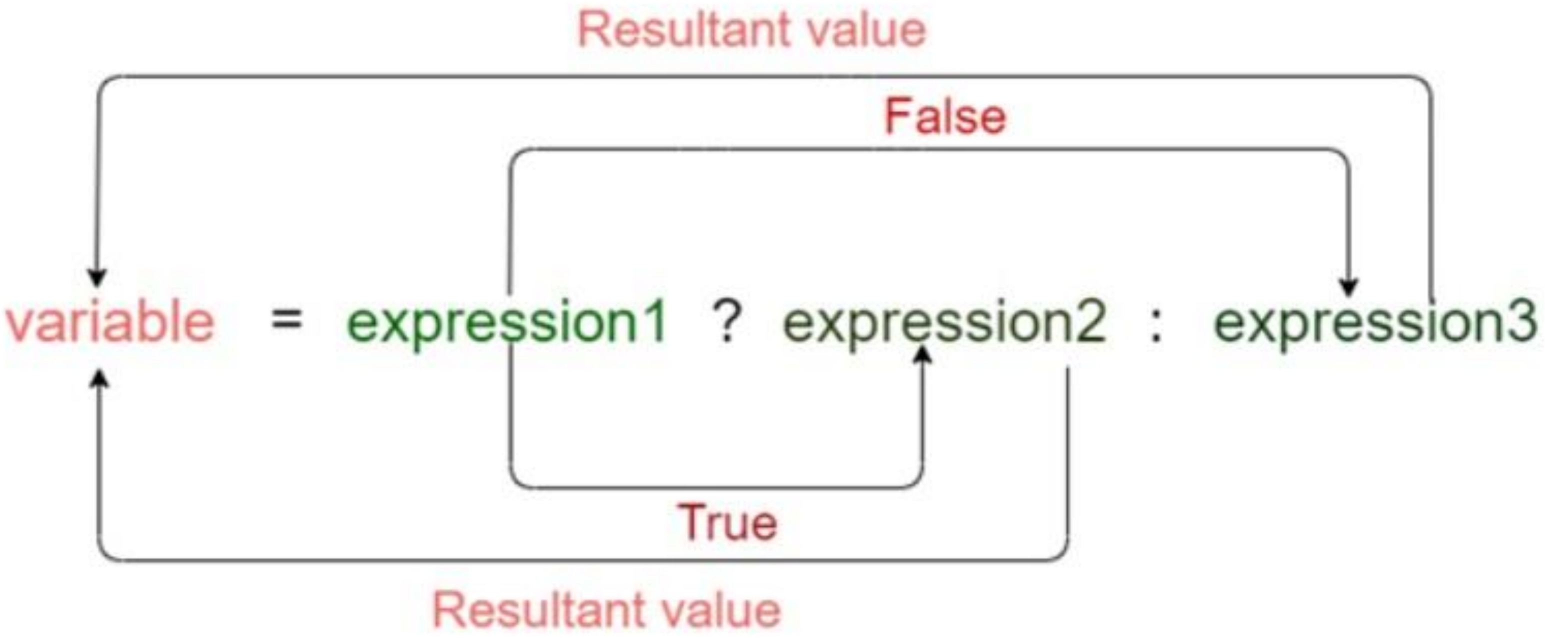
Conditional Operator (?:)

- Conditional operator (`? :`) takes three arguments
  - Ternary operator
- Syntax for using the conditional operator:

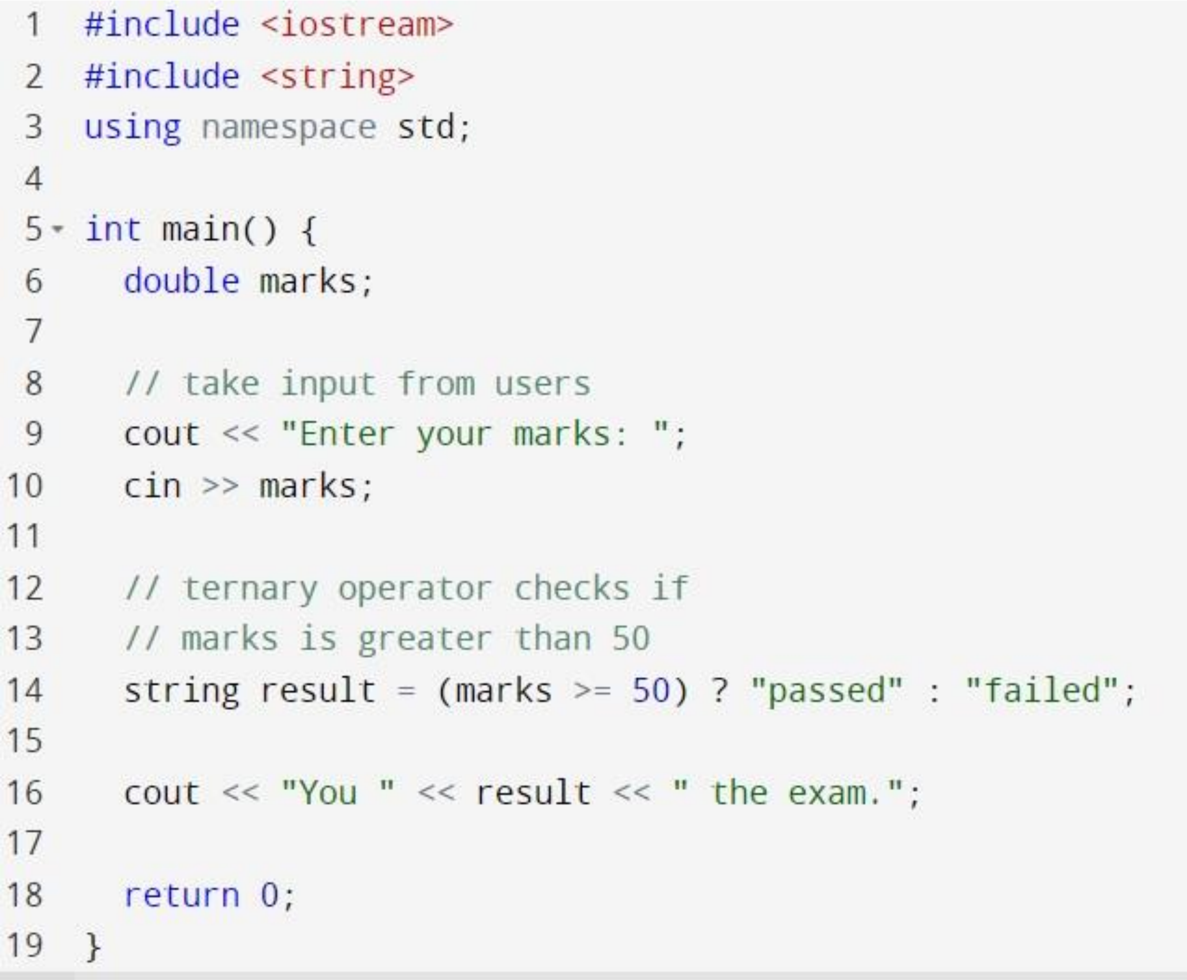
```
expression1 ? expression2 : expression3
```

- If `expression1` is `true`, the result of the conditional expression is `expression2`
  - Otherwise, the result is `expression3`

Conditional Operator (?:)



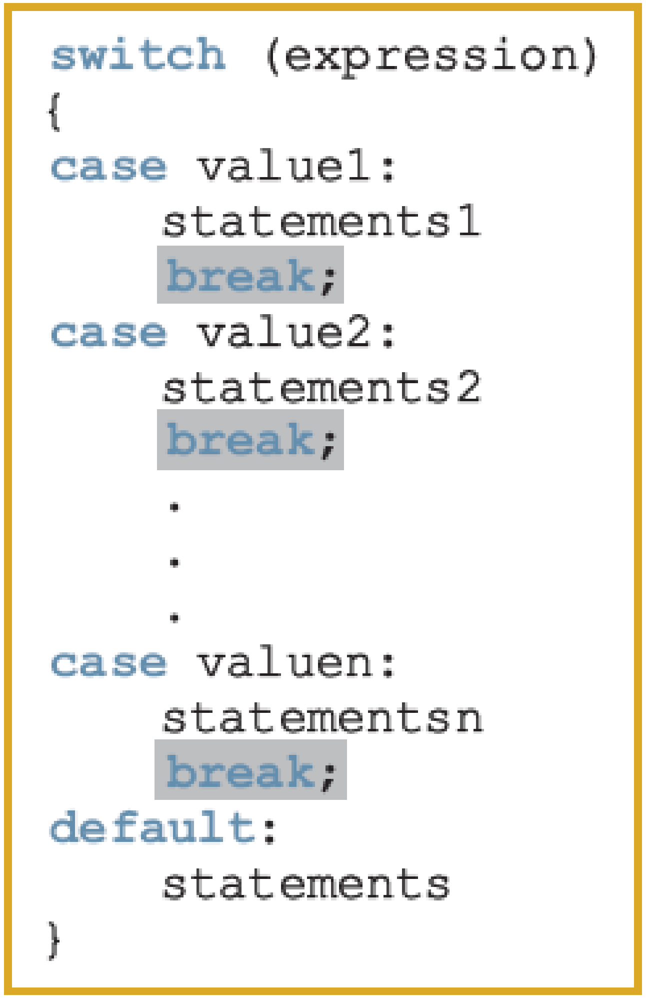
Example



```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
  double marks;

  // take input from users
  cout << "Enter your marks: ";
  cin >> marks;

  // ternary operator checks if
  // marks is greater than 50
  string result = (marks >= 50) ? "passed" : "failed";

  cout << "You " << result << " the exam.";

  return 0;
}
```

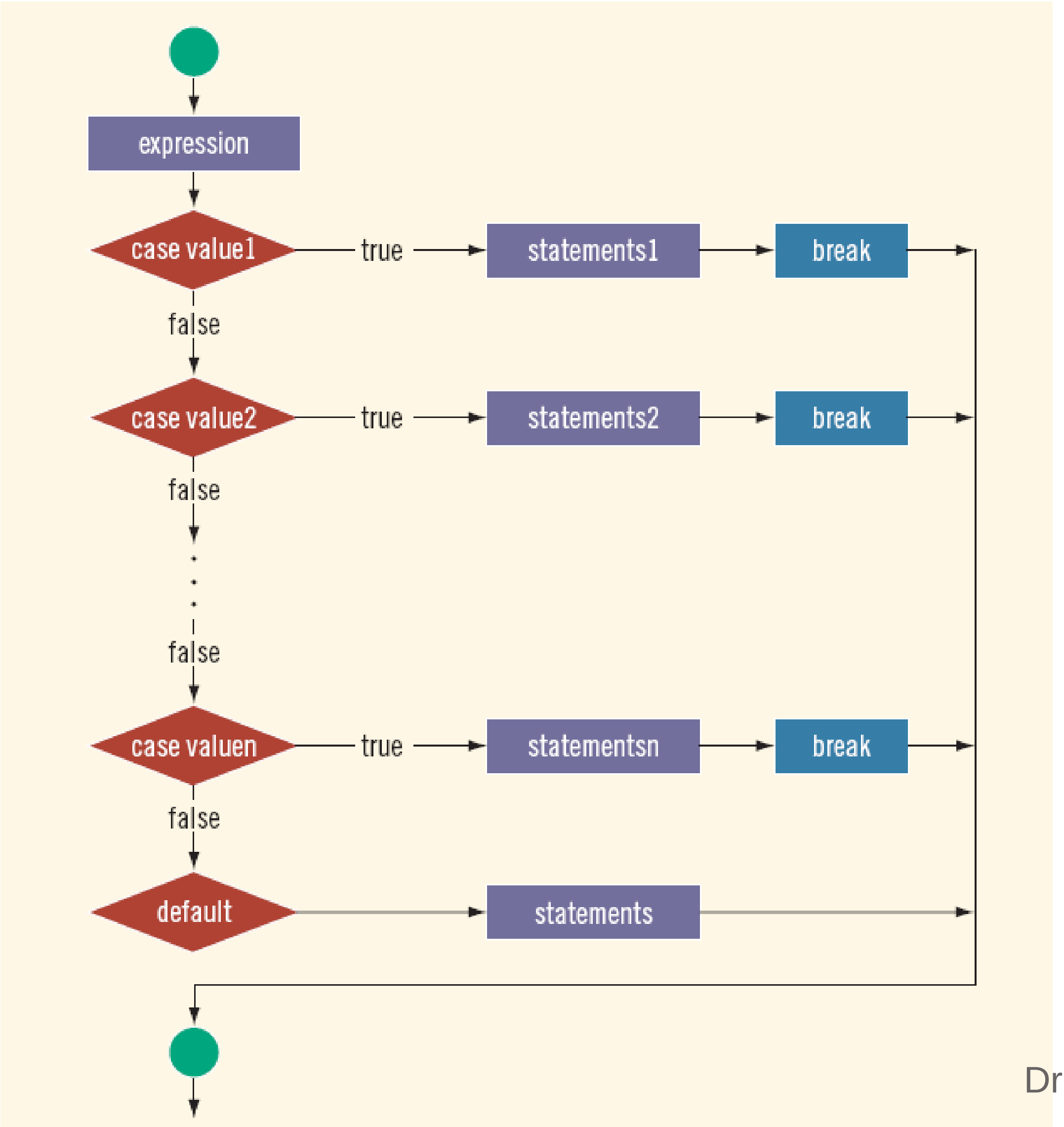
switch Structures

- `switch` structure: alternate to if-else
- `switch` (integral) expression is evaluated first
- Value of the expression determines which corresponding action is taken
- Expression is sometimes called the selector



```
switch (expression)
{
case value1:
    statements1
    break;
case value2:
    statements2
    break;
    .
    .
    .
case valuen:
    statementsn
    break;
default:
    statements
}
```

switch Structures



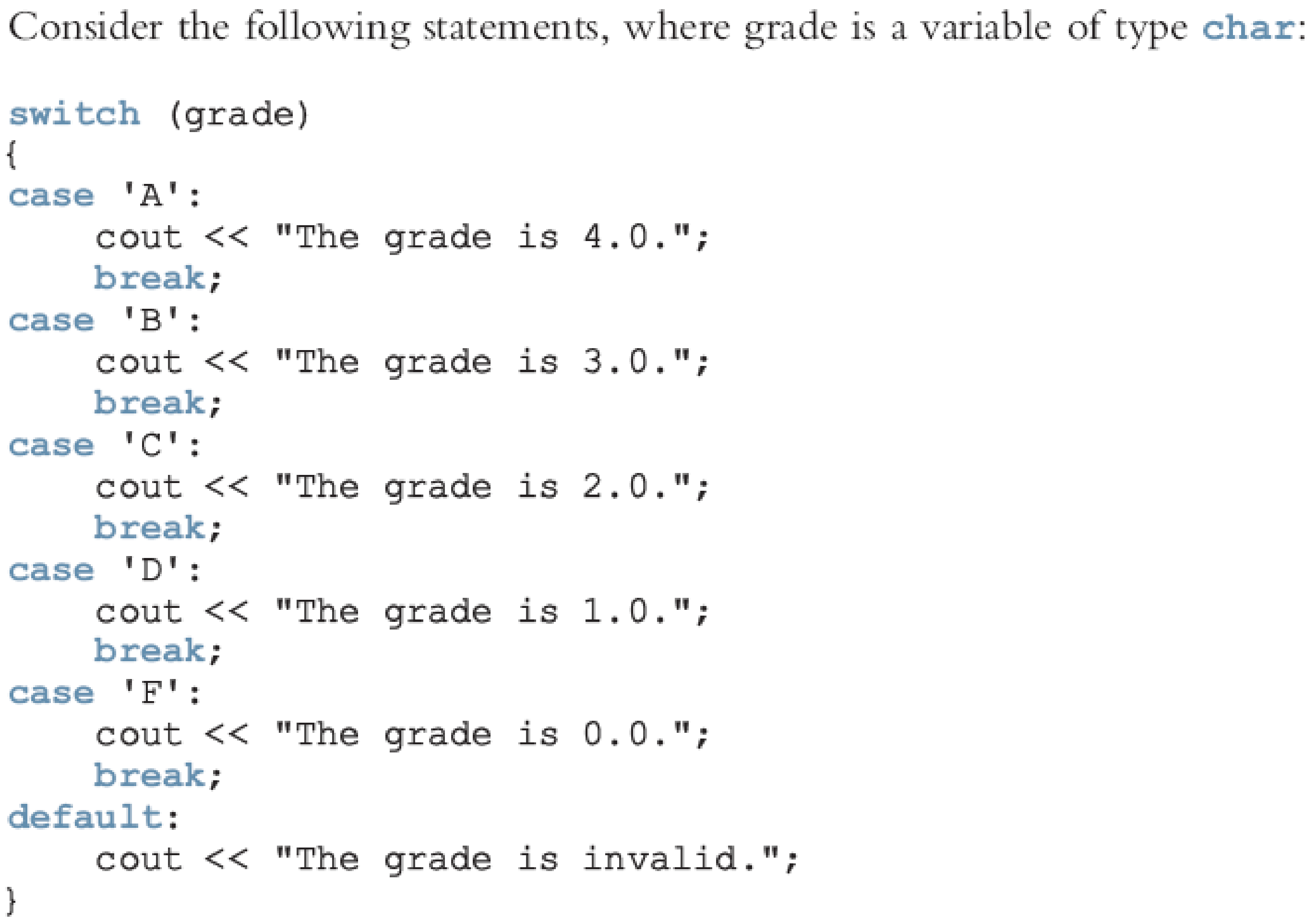
switch Structures

- One or more statements may follow a case label
- Braces are not needed to turn multiple statements into a single compound statement
- The `break` statement may or may not appear after each statement
- `switch`, `case`, `break`, and `default` are reserved words

Example

Write a program to input the grade in characters and print the equivalent GPA

Example



Consider the following statements, where grade is a variable of type char:

```
switch (grade)
{
case 'A':
    cout << "The grade is 4.0.";
    break;
case 'B':
    cout << "The grade is 3.0.";
    break;
case 'C':
    cout << "The grade is 2.0.";
    break;
case 'D':
    cout << "The grade is 1.0.";
    break;
case 'F':
    cout << "The grade is 0.0.";
    break;
default:
    cout << "The grade is invalid.";
}
```

Programming Example: Cable Company Billing

- This programming example calculates a customer's bill for a local cable company
- There are two types of customers:
  - Residential
  - Business
- Two rates for calculating a cable bill:
  - One for residential customers
  - One for business customers

Programming Example: Rates

- For residential customer:
  - Bill processing fee: $4.50
  - Basic service fee: $20.50
  - Premium channel: $7.50 per channel
- For business customer:
  - Bill processing fee: $15.00
  - Basic service fee: $75.00 for first 10 connections and $5.00 for each additional connection
  - Premium channel cost: $50.00 per channel for any number of connections

Programming Example: Requirements

- Ask user for account number and customer code
- Assume `R` or `r` stands for residential customer and `B` or `b` stands for business customer

Programming Example: Input and Output

- Input:
  - Customer account number
  - Customer code
  - Number of premium channels
  - For business customers, number of basic service connections
- Output:
  - Customer’s account number
  - Billing amount

Programming Example: Program Analysis

- Purpose: calculate and print billing amount
- Calculating billing amount requires:
  - Customer for whom the billing amount is calculated (residential or business)
  - Number of premium channels to which the customer subscribes
- For a business customer, you need:
  - Number of basic service connections
  - Number of premium channels

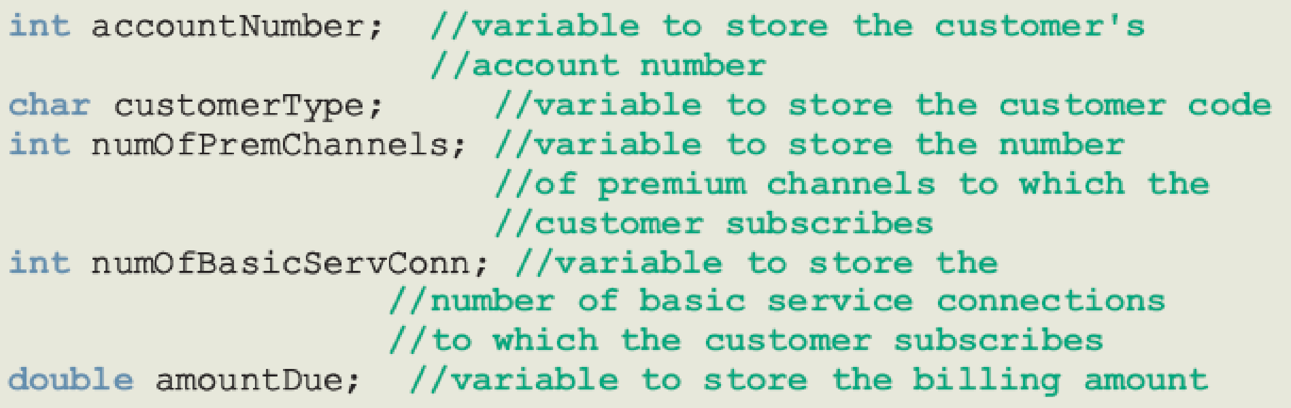
Programming Example: Program Analysis

- Data needed to calculate the bill, such as bill processing fees and the cost of a premium channel, are known quantities
- The program should print the billing amount to two decimal places

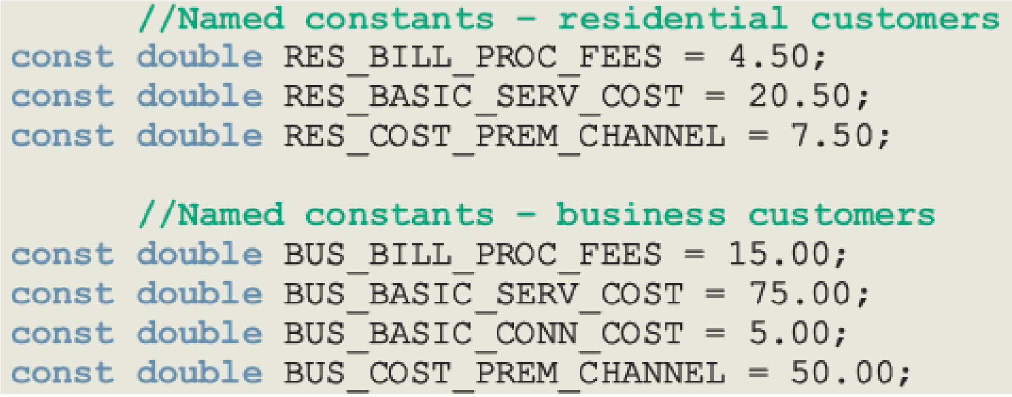
Programming Example: Algorithm Design

- Set precision to two decimal places
- Prompt user for account number and customer type
- If customer type is `R` or `r`
  - Prompt user for number of premium channels
  - Compute and print the bill
- If customer type is `B` or `b`
  - Prompt user for number of basic service connections and number of premium channels
  - Compute and print the bill

Programming Example: Variables and Named Constants



```
int accountNumber;  //variable to store the customer's
                    //account number
char customerType;     //variable to store the customer code
int numOfPremChannels; //variable to store the number
                       //of premium channels to which the
                       //customer subscribes
int numOfBasicServConn; //variable to store the
                  //number of basic service connections
                  //to which the customer subscribes
double amountDue;  //variable to store the billing amount
```



```
      //Named constants - residential customers
const double RES_BILL_PROC_FEES = 4.50;
const double RES_BASIC_SERV_COST = 20.50;
const double RES_COST_PREM_CHANNEL = 7.50;

      //Named constants - business customers
const double BUS_BILL_PROC_FEES = 15.00;
const double BUS_BASIC_SERV_COST = 75.00;
const double BUS_BASIC_CONN_COST = 5.00;
const double BUS_COST_PREM_CHANNEL = 50.00;
```

Programming Example: Formulas

Billing for residential customers:

```
amountDue = RES_BILL_PROC_FEES +
            RES_BASIC_SERV_COST
            + numOfPremChannels *
            RES_COST_PREM_CHANNEL;
```

Programming Example: Formulas

Billing for business customers:

```
if (numOfBasicServConn <= 10)
    amountDue = BUS_BILL_PROC_FEES +
                BUS_BASIC_SERV_COST
                + numOfPremChannels *
                  BUS_COST_PREM_CHANNEL;
else
    amountDue = BUS_BILL_PROC_FEES +
                BUS_BASIC_SERV_COST
                + (numOfBasicServConn - 10)
                 * BUS_BASIC_CONN_COST
               + numOfPremChannels *
                 BUS_COST_PREM_CHANNEL;
```

Programming Example: Main Algorithm

1. Output floating-point numbers in fixed decimal with decimal point and trailing zeros
   - Output floating-point numbers with two decimal places and set the precision to two decimal places
2. Prompt user to enter account number
3. Get customer account number
4. Prompt user to enter customer code
5. Get customer code

Programming Example: Main Algorithm

6. If the customer code is `r` or `R`,
   - Prompt user to enter number of premium channels
   - Get the number of premium channels
   - Calculate the billing amount
   - Print account number and billing amount

Programming Example: Main Algorithm

7. If customer code is b or B,
    - Prompt user to enter number of basic service connections
    - Get number of basic service connections
    - Prompt user to enter number of premium channels
    - Get number of premium channels
    - Calculate billing amount
    - Print account number and billing amount

Programming Example: Main Algorithm

8. If customer code is other than `r`, `R`, `b`, or `B`, output an error message

Thanks!