Chapter 3 - Branches

Section 3.1 - If-else

Like a river splitting and re-merging, ***branching*** directs a program to execute either one statement group or another, depending on an expression's value. An example is to print "Too young to drive" if userAge < 16, else print "OK to drive". The language's if-else statement supports branching.

Construct 3.1.1: If-else statement.

```
// Statements that execute before the branches

if (expression) {
   // Statements to execute when the expression is true (first branch)
}
else {
   // Statements to execute when the expression is false (second branch)
}

// Statements that execute after the branches
```

Figure 3.1.1: If-else example: Car insurance prices.

```
#include <stdio.h>

int main(void) {
   const int PRICE_LESS_THAN_25 = 4800; // Age less than 25
   const int PRICE_25_AND_UP    = 2200; // Age 25 and Up
   int userAge                  = 0;    // Years
   int insurancePrice           = 0;    // Dollars

   printf("Enter age: ");
   scanf("%d", &userAge);

   if (userAge < 25) {
      insurancePrice = PRICE_LESS_THAN_25;
      printf("(executed first branch)\n");
   }
   else {
      insurancePrice = PRICE_25_AND_UP;
      printf("(executed second branch)\n");
   }

   printf("Annual price: $%d\n", insurancePrice);

   return 0;
}
```

```
Enter age: 19
(executed first bran
Annual price: $4800

...

Enter age: 28
(executed second bra
Annual price: $2200
```

If a user inputs an age less than 25, the statement `insurancePrice = PRICE_LESS_THAN_25` executes. Otherwise, `insurancePrice = PRICE_25_AND_UP` executes. (Prices under 25 are higher because 1 in 6 such drivers are involved in an accident each year, vs. 1 in 15 for older drivers. Source: www.census.gov, 2009).

Though not required, programmers follow the good practice of indenting a branch's statements, using a consistent number of spaces. This material indents 3 spaces.

P Participation Activity

3.1.1: An if-else is like a branching road.

P Participation Activity

3.1.2: If-else statements.

#	Question	Your answer

1	What is the final value of numItems?
	`bonusVal = 5;`
	`if (bonusVal < 12) {`
	`   numItems = 100;`
	`}`
	`else {`
	`   numItems = 200;`
	`}`	

2	What is the final value of numItems?
	`bonusVal = 12;`
	`if (bonusVal < 12) {`
	`   numItems = 100;`
	`}`
	`else {`
	`   numItems = 200;`
	`}`	

3

What is the final value of numItems?

```
bonusVal = 15;
numItems = 44;
if (bonusVal < 12) {
   numItems = numItems + 3;
}
else {
   numItems = numItems + 6;
}
numItems = numItems + 1;
```

4

What is the final value of bonusVal?

```
bonusVal = 11;
if (bonusVal < 12) {
   bonusVal = bonusVal + 2;
}
else {
   bonusVal = bonusVal + 10;
}
```

5

What is the final value of bonusVal?

```
bonusVal = 11;
if (bonusVal < 12) {
   bonusVal = bonusVal + 2;
   bonusVal = 3 * bonusVal;
}
else {
   bonusVal = bonusVal + 10;
}
```

P Participation Activity

3.1.3: Writing an if-else statement.

Translate each description to an if-else statement as directly as possible. Use { }. (Not checked, but please indent a branch's statements some consistent number of spaces such as 3 spaces).

#	Question	Your answer
1	If userAge is greater than 62, assign 15 to discount. Else, assign 0 to discount.	
2	If numPeople is greater than 10, execute groupSize = 2 * groupSize. Otherwise, execute groupSize = 3 * groupSize and also numPeople = numPeople - 1.	
3	If numPlayers is greater than 11, execute teamSize = 11. Otherwise, execute teamSize = numPlayers. Then, no matter the value of numPlayers, execute teamSize = 2 * teamSize.	

An if statement can be written without the else part. Such a statement acts like an if-else with no statements in the else branch.

Figure 3.1.2: If statement without else: Absolute value example.

```c
#include <stdio.h>

int main(void) {
   int userVal = 0;
   int absVal  = 0;

   printf("Enter an integer: ");
   scanf("%d", &userVal);

   absVal = userVal;
   if (absVal < 0) {
      absVal = absVal * -1;
   }

   printf("The absolute value of %d", userVal);
   printf(" is %d\n", absVal);

   return 0;
}
```

```
Enter an integer: -55
The absolute value of -55 is 55

...

Enter an integer: 42
The absolute value of 42 is 42
```

(The example used the number 42. That's a popular number. Just for fun, search for "the answer to life the universe and everything" on Google to learn why).

P Participation Activity

3.1.4: If without else.

What is the final value of numItems?

#	Question	Your answer

1	`bonusVal = 19;`
	`numItems = 1;`
	`if (bonusVal > 10) {`
	`   numItems = numItems + 3;`
	`}`	

2	`bonusVal = 0;`
	`numItems = 1;`
	`if (bonusVal > 10) {`
	`   numItems = numItems + 3;`
	`}`	

Braces surround a branch's statements. ***Braces*** { }, sometimes redundantly called curly braces, represent a grouping, such as a grouping of statements. Note: { } are braces, [ ] are brackets.

When a branch has a single statement, the braces are optional, but good practice *always* uses the braces. Always using braces even when a branch only has one statement prevents the common error of mistakenly thinking a statement is part of a branch.

P Participation Activity

3.1.5: Leaving off braces can lead to a common error; better to always use braces.

P Participation Activity

3.1.6: Omitting braces is a common source of errors.

What is the final value of numItems?

#	Question	Your answer

1	`numItems = 0;`
	`bonusVal = 19;`
	`if (bonusVal > 10)`
	`   numItems = bonusVal;`
	`numItems = numItems + 1;`	

2	`numItems = 0;`
	`bonusVal = 5;`
	`if (bonusVal > 10)`
	`   // Need to update bonusVal`
	`   numItems = bonusVal;`
	`numItems = numItems + 1;`	

3	`numItems = 0;`
	`bonusVal = 5;`
	`if (bonusVal > 10)`
	`   // Update bonusVal`
	`   bonusVal = bonusVal - 1;`
	`   numItems = bonusVal;`
	`numItems = numItems + 1;`	

C Challenge Activity

3.1.1: Enter the output for the if-else branches.

C Challenge Activity

3.1.2: Basic if-else expression.

C Challenge Activity 3.1.3: Basic if-else.

Section 3.2 - Relational and equality operators

An if-else expression commonly involves a ***relational operator*** or ***equality operator***.

Table 3.2.1: Relational (first four) and equality (last two) operators.

Relational and equality operators	Description
a **<** b	a is ***less-than*** b
a **>** b	a is ***greater-than*** b
a **<=** b	a is ***less-than-or-equal-to*** b
a **>=** b	a is ***greater-than-or-equal-to*** b
a **==** b	a is ***equal to*** b
a **!=** b	a is ***not equal to*** b

Each operator involves two operands, shown above as a and b. The operation evaluates to a ***Boolean*** value meaning either *true* or *false*. If userAge is 19, then userAge < 25 evaluates to true.

Some operators like >= involve two characters. Only the shown two-character sequences represent valid operators. A common error is to use invalid character sequences like =>, !<, or <>, which are *not* valid operators.

Note that equality is ==, not =.

P Participation Activity

3.2.1: Expressions with relational and equality operators.

Type the operator to complete the desired expression.

```
if expression  {
   ...
}
else {
   ...
}
```

#	Question	Your answer
1	numDogs is 0	(numDogs □ 0)
2	numDogs is greater than 10	(numDogs □ 10)
3	numCars is greater than or equal to 5	(numCars □ 5)
4	numCars is 5 or greater	(numCars □ 5)
5	numDogs and numCats are the same	(numDogs □ numCats)
6	numDogs and numCats differ	(numDogs □ numCats)
7	numDogs is either less-than or greater-than numCats	(numDogs □ numCats)
8	centsLost is a negative number	(centsLost □ 0)
9	userChar is the character 'x'.	(userChar □ 'x')

P Participation Activity

3.2.2: If-else with expression: Non-negative.

The program prints "Zero" if the user enters 0, else prints "Non-zero". Modify the program to print "Non-negative" if the user enters 0 or greater, else print "Negative".

The relational and equality operators work for integer, character, and floating-point built-in types. Comparing characters compares their ASCII numerical encoding. However, floating-point types should not be compared using the equality operators, due to the imprecise representation of floating-point numbers, as discussed in a later section.

The operators should not be used with strings; unexpected results will occur. See another section discussing the string comparison function strcmp().

Perhaps the most common error in C and C++ is to use = rather than == in an if-else expression, as in: if (numDogs = 9) { ... }. That is not a syntax error. The statement assigns 9 to numDogs, and then because that value is non-zero, the expression is considered true. C's designers allowed assignment in expressions to allow compact code, and use = for assignment rather than := or similar to save typing. Many people believe those language design decisions were mistakes, leading to many bugs. Some modern compilers provide a warning when = appears in an if-else expression.

P Participation Activity

3.2.3: Watch out for assignment in an if-else expression.

What is the final value of numItems?

#	Question	Your answer

1	`numItems = 3;`
	`if (numItems == 3) {`
	`   numItems = numItems + 1;`
	`}`	

2	`numItems = 3;`
	`if (numItems = 10) {`
	`   numItems = numItems + 1;`
	`}`	

P Participation Activity

3.2.4: Comparing various types.

Which comparison will compile AND consistently yield expected results? Variables have types denoted by their names.

#	Question	Your answer
1	myInt == 42	OK Not OK
2	myChar == 'q'	OK Not OK
3	myDouble == 3.25	OK Not OK

P Participation Activity

3.2.5: Comparing various types (continued).

#	Question	Your answer
1	myString == "Hello"	OK Not OK

C Challenge Activity

3.2.1: Enter the output for the branches with relational operators.

C Challenge Activity

3.2.2: If-else expression: Detect greater than 100.

C Challenge Activity

3.2.3: Basic If-else expression: Detect odd.

C Challenge Activity

3.2.4: If-else statement: Fix errors.

C Challenge Activity

3.2.5: If-else statement: Print senior citizen.

Section 3.3 - Multiple if-else branches

Commonly, a programmer requires more than two branches, in which case a multi-branch if-else arrangement can be used.

Construct 3.3.1: Multi-branch if-else arrangement. Only 1 branch will execute.

```
if (expr1) {

}
else if (expr2) {

}

...

else if (exprN) {

}
else {

}
```

Figure 3.3.1: Multiple if-else branches example: Anniversaries.

```
#include <stdio.h>

int main(void) {
   int numYears = 0;

   printf("Enter number years married: ");
   scanf("%d", &numYears);

   if (numYears == 1) {
      printf("Your first year -- great!\n");
   }
   else if (numYears == 10) {
      printf("A whole decade -- impressive.\n");
   }
   else if (numYears == 25) {
      printf("Your silver anniversary -- enjoy.\n");
   }
   else if (numYears == 50) {
      printf("Your golden anniversary -- amazing.\n");
   }
   else {
      printf("Nothing special.\n");
   }

   return 0;
}
```

```
Enter number years marrie
A whole decade -- impress

...

Enter number years marrie
Your silver anniversary -

...

Enter number years marrie
Nothing special.

...

Enter number years marrie
Your first year -- great!
```

P Participation Activity

3.3.1: Only one branch will execute in a multi-branch if-else arrangement.

P Participation Activity

3.3.2: Multi-branch if-else.

What is the final value of employeeBonus for each given value of numSales?

```
if (numSales == 0) {
   employeeBonus = 0;
}
else if (numSales == 1) {
   employeeBonus = 2;
}
else if (numSales == 2) {
   employeeBonus = 5;
}
else {
   employeeBonus = 10;
}
```

#	Question	Your answer
1	numSales is 2	
2	numSales is 0	
3	numSales is 7	

P Participation Activity

3.3.3: Complete the multi-branch if-else.

```
if (userChar == 'x') {     // User typed x
   numTries = 3;
}
__________________________ // User typed y
   numTries = 7;
}
else {
   numTries = 1;
}
```

#	Question	Your answer
1	Fill in the missing line of code.	

Programmers commonly use the sequential nature of the multi-branch if-else arrangement to detect ranges of numbers. In the following example, the second branch expression is only reached if the first expression is false. So the second branch is taken if userAge is *NOT* <= 15 (meaning 16 or greater) AND userAge is <=24, meaning userAge is between 16..24 (inclusive).

Figure 3.3.2: Using sequential nature of multi-branch if-else for ranges: Insurance prices.

```
#include <stdio.h>

int main(void) {
   const int PRICE_16_TO_24  = 4800; // Age 16..24 (2010 U.S., carsdirect.com)
   const int PRICE_25_TO_39  = 2350; // Age 25..39
   const int PRICE_40_AND_UP = 2100; // Age 40 and up
   int userAge        = 0;
   int insurancePrice = 0;

   printf("Enter your age: ");
   scanf("%d", &userAge);

   if (userAge <= 15) {                 // Age 15 and under
      printf("Too young.\n");
      insurancePrice = 0;
   }
   else if (userAge <= 24) {            // Age 16..24
      insurancePrice = PRICE_16_TO_24;
   }
   else if (userAge <= 39) {            // Age 25..39
      insurancePrice = PRICE_25_TO_39;
   }
   else {                               // Age 40 and up
      insurancePrice = PRICE_40_AND_UP;
   }

   printf("Annual price: $%d\n", insurancePrice);

   return 0;
}
```

P Participation Activity

3.3.4: Ranges and multi-branch if-else.

Type the range for each branch, typing 10..13 to represent range 10, 11, 12, 13, and typing 10+ to represent all numbers 10 and larger.

```
if (numSales <= 9) {
   ...
}
else if (numSales <= 19) { // 2nd branch range: _______
   ...
}
else if (numSales <= 29) { // 3rd branch range: _______
   ...
}
else {                     // 4th branch range: _______
   ...
}
```

#	Question	Your answer
1	2nd branch range:	
2	3rd branch range:	
3	4th branch range:	

4	What is the range for the last branch below?
	`if (numItems < 0) {`
	`   ...`
	`}`
	`else if (numItems > 100) {`
	`   ...`
	`}`
	`else {   // Range: ______`
	`   ...`
	`}`	

P Participation Activity

3.3.5: Complete the multi-branch code.

#	Question	Your answer
	Second branch:	`if (userNum < 100 ) {`

1

Second branch: userNum is less than 200

```
if (userNum < 100 ) {
   ...
}
else if ( □ ) {
  ...
}
else { // userNum >= 200
   ...
}
```

2

Second branch: userNum is positive (non-zero)

```
if (userNum < 0 ) {
   ...
} □ {
  ...
}
else { // userNum is 0
   ...
}
```

3

Second branch: userNum is greater than 105

```
if (userNum < 100 ) {
   ...
} □ {
  ...
}
else { // userNum is between
       // 100 and 105
   ...
}
```

4

If the final else branch executes, what must userNum have been? Type "unknown" if appropriate.

```
if (userNum <= 9) {
   ...
}
else if (userNum >=
11) {
   ...
}
else {
   ... // userNum if
this executes?
}
```

□

Which branch will execute? Valid answers: 1, 2, 3, or none.

```
userNum = 555;
if (userNum < 0) {
   ... // Branch 1
```

□

```
}
else if (userNum ==
0) {
   ... // Branch 2
}
else if (userNum <
100) {
   ... // Branch 3
}
```

A branch's statements can include any valid statements, including another if-else statement, such occurrence known as ***nested if-else*** statements.

Figure 3.3.3: Nested if-else.

```
if (userChar == 'q') { // userChar 'q'
   ...
}
else if (userChar == 'c') {
   if (numItems < 0) { // userChar 'c' and numItems < 0
      ...
   }
   else {                // userChar 'c' and numItems >= 0
      ...
   }
}
else { // userChar not 'q' or 'c'
   ...
}
```

Sometimes the programmer has multiple if statements in sequence, which looks similar to a multi-branch if-else statement but has a very different meaning. Each if-statement is independent, and thus more than one branch can execute, in contrast to the multi-branch if-else arrangement.

Figure 3.3.4: Multiple distinct if statements.

```
#include <stdio.h>

int main(void) {
   int userAge = 0;

   printf("Enter age: ");
   scanf("%d", &userAge);

   // Note that more than one "if" statement can execute
   if (userAge < 16) {
      printf("Enjoy your early years.\n");
   }

   if (userAge >= 16) {
      printf("You are old enough to drive.\n");
   }

   if (userAge >= 18) {
      printf("You are old enough to vote.\n");
   }

   if (userAge >= 25) {
      printf("Most car rental companies will rent to you.\n");
   }

   if (userAge >= 35) {
      printf("You can run for president.\n");
   }

   return 0;
}
```

```
Enter age: 12
Enjoy your early

...

Enter age: 27
You are old enoug
You are old enoug
Most car rental c

...

Enter age: 99
You are old enoug
You are old enoug
Most car rental c
You can run for p
```

P Participation Activity

3.3.6: Multiple if statements.

P Participation Activity

3.3.7: If statements.

Determine the final value of numBoxes.

#	Question	Your answer

1	`numBoxes   = 0;`
	`numApples = 9;`
	`if (numApples < 10) {`
	`   numBoxes = 2;`
	`}`
	`if (numApples < 20) {`
	`   numBoxes = numBoxes + 1;`
	`}`	

2	`numBoxes   = 0;`
	`numApples = 9;`
	`if (numApples < 10) {`
	`   if (numApples < 5) {`
	`      numBoxes = 1;`
	`   }`
	`   else {`
	`      numBoxes = 2;`
	`   }`
	`}`
	`else if (numApples < 20) {`
	`   numBoxes = numBoxes + 1;`
	`}`	

C Challenge Activity

3.3.1: Enter the output for the multiple if-else branches.

C Challenge Activity

3.3.2: If-else statement: Fix errors.

C Challenge Activity

3.3.3: Multiple branch If-else statement: Print century.

C Challenge Activity

3.3.4: Multiple if statements: Print car info.

Section 3.4 - Logical operators

More operators are available for use in expressions. A ***logical operator*** treats operands as being true or false, and evaluates to true or false.

Table 3.4.1: Logical operators.

Logical operator	Description
a **&&** b	Logical AND: true when *both* of its operands are true
a **||** b	Logical OR: true when *at least one* of its two operands are true
!a	Logical NOT (opposite): true when its single operand is false (and false when operand is true)

The operands, shown above as a and b, are typically expressions.

Table 3.4.2: Logical operators examples.

Given age = 19, days = 7, userChar = 'q'	
`(age > 16) && (age < 25)`	true, because both operands are true.
`(age > 16) && (days > 10)`	false, because both operands are not true (days > 10 is false).
`(age > 16) || (days > 10)`	true, because at least one operand is true (age > 16 is true).
`!(days > 10)`	true, because operand is false.
`!(age > 16)`	false, because operand is true.
`!(userChar == 'q')`	false, because operand is true.

P Participation Activity

3.4.1: Evaluating expressions with logical operators.

Given numPeople = 10, numCars = 2, userKey = 'q'.

#	Question
1	`numPeople >= 10`
2	`(numPeople >= 10) && (numCars > 2)`
3	`(numPeople >= 20) || (numCars > 1)`
4	`!(numCars < 5)`
5	`!(userKey == 'a')`

5	
6	`userKey != 'a'`
7	`!((numPeople > 10) && (numCars > 2))`
8	`(userKey == 'x') || ((numPeople > 5) && (numCars > 1))`

P Participation Activity

3.4.2: Logical operators: Complete the expressions for the given condition.

#	Question	Your answer

1	days is greater than 30 and less than 90	`if ( (days > 30)` □ `(days < 90) ) {`
		`   ...`
		`}`

2	0 < maxCars < 100	`if ( (maxCars > 0)` □ `(maxCars < 100) ) {`
		`   ...`
		`}`

3	numStores is between 10 and 20, inclusive.	`if ( (numStores >= 10) && (` □
		`   ...`
		`}`

4	numDogs is 3 or more and numCats is 3 or more	`if ( (numDogs >= 3)` □ `) {`
		`   ...`
		`}`

	3 or more.	

5	Either wage is greater than 10 or age is less than 18. Use ||. Use > and < (not >= and <=). Use parentheses around sub-expressions.	`if (  ) {`
		`   ...`
		`}`

6	num is a 3-digit positive integer, such as 100, 989, or 523, but not 55, 1000, or -4.
	For most direct readability, your expression should compare directly with the smallest and largest 3-digit number.
		`if ( (num >= 100)  )`
		`   ...`
		`}`

P Participation Activity

3.4.3: Indicate which are correct expressions for the desired conditions.

#	Question	Your answer
1	userNum is less than -5 or greater than 10: `(userNum < -5) && (userNum > 10)`	Correct Incorrect
2	userNum is not greater than 100: `(userNum !> 100)`	Correct Incorrect
3	userNum is neither 5 nor 10: `!( (userNum == 5) || (userNum == 10) )`	Correct Incorrect
4	userNum is between 10 and 20, inclusive `( (userNum >= 10) || (userNum <= 20) )`	Correct Incorrect

The ***bool*** (short for Boolean) data type is for variables that should store only values true or false. Thus, a programmer can define a variable like `bool result;`. The programmer can assign the variable as in `result = true`, or as in `result = (age < 25)`, or as in `result = x && y;`. The programmer can use the variable in an if-else statement as in if (result) or as in `if ((!result) && (b == c))`.

Note: the implementation of true/false values is somewhat inelegant. false is actually 0, and true is 1, and any non-zero value in an expression is considered true also.

A common error often made by new programmers is to write expressions like `if (16 < age < 25)`, as one might see in mathematics.

The meaning, however, almost certainly is not what the programmer intended. Suppose age is presently 28. The expression is evaluated left-to-right, so evaluation of `16 < age` yields true. Next, the expression `true < 25` is evaluated; clearly not the programmer's intent. However, as mentioned above, true is actually 1, and evaluating `1 < 25` will yield true. Thus, for any age greater than 16, the above expression evaluates to true, even for ages greater than 25. The key is to note two things:

1. The relational operators and logical operators (except for !) are binary operators. ***Binary operators*** take two operands (from the left and right) and evaluate to true or false.
2. Only one operator is evaluated at a time, based on precedence rules.

Based on those key points, note that `16 < age < 25` is actually the same as `(16 < age) < 25`, which evaluates to `(true) < 25` for any age over 16, which is the same as `(1) < 25`, which evaluates to true. Recall that the correct way to do the comparison is: `(age > 16) && (age < 25)`.

Logical, relational, and bitwise expressions are evaluated using precedence rules:

Table 3.4.3: Precedence rules for logical and relational operators.

Convention	Description	Explanation
()	Items within parentheses are evaluated first.	In `!(age > 16)`, age > 16 is evaluated first, then the logical NOT.
!	Next to be evaluated is ***!***.	
**** / % + -***	Arithmetic operator are then evaluated using the precedence rules for those operators.	`z - 45 < 53` is evaluated as `(z - 45) < 53`.
< <= > >=	Then, relational operators **< <= > >=** are evaluated.	`x < 2 || x >= 10` is evaluated as `(x < 2) || (x >= 10)` because < and >= have precedence over ||.
== *!=*	Then, the equality and inequality operators **== *!=*** are evaluated.	`x == 0 && x >= 10` is evaluated as `(x == 0) && (x >= 10)` because < and >= have precedence over &&.
&	Then, the bitwise AND operator is evaluated.	`x == 5 | y == 10 & z != 10` is evaluated as `(x == 5) | ((y == 10) & (z != 10))` because & has precedence over |.
|	Then, the bitwise OR operator is evaluated.	`x == 5 | y == 10 && z != 10` is evaluated as `((x == 5) | (y == 10)) && (z != 10))` because | has precedence over &&.
&&	Then, the logical AND operator is evaluated.	`x == 5 || y == 10 && z != 10` is evaluated as `(x == 5) || ((y == 10) && (z != 10))` because && has precedence over ||.
||	Finally, the logical OR operator is evaluated.	

P Participation Activity

3.4.4: Logical expression simulator.

Try typing different expressions involving x, y and observe whether the expression evaluates to true.

Using parentheses makes the order of evaluation explicit, rather than relying on precedence rules. Thus, `(age > 16) || (age < 25)` is preferable over `age > 16 || age < 25`, even though both expressions evaluate the same because > and < have higher precedence than ||.

Using parentheses to make order of evaluation explicit becomes even more critical as arithmetic, relational, equality, and logical operators are combined in a single expression. For example, a programmer might write:

- `! x == 2` intending to mean `!(x == 2)`, but in fact the compiler computes `(!x) == 2` because ! has precedence over ==.
- `w && x  == y && z` intending `(w && x) == (y && z)`, but the compiler computes `(w && (x == y)) && z` because == has precedence over &&.
- `! x + y < 5` intending `!((x + y) < 5)`, but the compiler computes `((!x) + y) < 5` because ! has precedence over +.

Good practice is to use parentheses in expressions to make the intended order of evaluation explicit.

P Participation Activity

3.4.5: Order of evaluation.

Which of the following expressions illustrate the correct order of evaluation with parentheses?

#	Question	Your answer
1	`! green == red`	(!green) == red
		!(green == red)
		(!green =)= red
	`bats < birds || birds < insects`	((bats < birds) || birds) < insects

<table>
<tr><td rowspan="2">2</td><td rowspan="2"></td><td>bats < (birds || birds) < insects</td></tr>
<tr><td>(bats < birds) || (birds < insects)</td></tr>
<tr><td rowspan="3">3</td><td rowspan="3"><code>! (bats < birds) || (birds < insects)</code></td><td>! ((bats < birds) || (birds < insects))</td></tr>
<tr><td>(! (bats < birds)) || (birds < insects)</td></tr>
<tr><td>((!bats) < birds) || (birds < insects)</td></tr>
<tr><td rowspan="3">4</td><td rowspan="3"><code>(num1 == 9) || (num2 == 0) && (num3 == 0)</code></td><td>(num1 == 9) || ((num2 == 0) && (num3 == 0))</td></tr>
<tr><td>((num1 == 9) || (num2 == 0)) && (num3 == 0)</td></tr>
<tr><td>(num1 == 9) || (num2 == (0 && num3) == 0)</td></tr>
</table>

The reader should note that the logical AND is && and not just &, and likewise that logical OR is || and not just |. The single character versions represent different operators known as ***bitwise*** operators, which perform AND or OR on corresponding individual bits of the operands.

Using bitwise operators when intending to use logical operators may yield different behavior than expected. A common error occurs when bitwise operators are used instead of logical operators by mistake.

P Participation Activity

3.4.6: Mixing bitwise and logical operators.

#	Question	Your answer
1	`x == 3 | y > 1 && z != 3` Which of the following expressions illustrates the correct order of evaluation with parentheses?	(x == 3) | ((y > 1) && (z != 3)) ((x == 3) | (y > 1)) && (z != 3)
2	`x == 3 & y > 1 || z != 3` Which of the following expressions illustrates the correct order of evaluation with parentheses?	((x == 3) & (y > 1)) || (z != 3) (x == 3) & ((y > 1) || (z != 3))

3	`x < 7 | y >= 10 && z == 15`
	For which values of x, y, and z does the expression evaluate to true?
		x = 4, y = 11, and z = 10
		x = 4, y = 11, and z = 15

C Challenge Activity

3.4.1: Detect specific values.

C Challenge Activity

3.4.2: Detect number range.

Section 3.5 - Switch statements

A ***switch*** statement can more clearly represent multi-branch behavior involving a variable being compared to constant values. The program executes the first ***case*** whose constant expression matches the value of the switch expression, executes that case's statements, and then jumps to the end. If no case matches, then the ***default case*** statements are executed.

Figure 3.5.1: Switch example: Estimates a dog's age in human years.

```
#include <stdio.h>

/* Estimates dog's age in equivalent human years.
   Source: www.dogyears.com
*/

int main(void) {
   int dogAgeYears = 0;

   printf("Enter your dog's age (in years): ");
   scanf("%d", &dogAgeYears);

   switch (dogAgeYears) {
      case 0:
         printf("That's 0..14 human years.\n");
         break;

      case 1:
         printf("That's 15 human years.\n");
         break;

      case 2:
         printf("That's 24 human years.\n");
         break;

      case 3:
         printf("That's 28 human years.\n");
         break;

      case 4:
         printf("That's 32 human years.\n");
         break;

      case 5:
         printf("That's 37 human years.\n");
         break;

      default:
         printf("Human years unknown.\n");
         break;
   }

   return 0;
}
```

```
Enter your dog's age (in years
That's 32 human years.

...

Enter your dog's age (in years
Human years unknown.
```

P Participation Activity

3.5.1: Switch statement.

A switch statement can be written using a multi-branch if-else statement, but the switch statement may make the programmer's intent clearer.

Figure 3.5.2: A switch statement may be clearer than an multi-branch if-else.

```
if (dogYears == 0) {           // Like case 0
   // Print 0..14 years
}
else if (dogYears == 1) {      // Like case 1
   // Print 15 years
}
...
else if (dogYears == 5) {      // Like case 5
   // Print 37 years
}
else {                        // Like default case
   // Print unknown
}
```

P Participation Activity

3.5.2: Switch statement.

numItems and userVal are int types. What is the final value of numItems for each userVal?

```
switch (userVal) {
   case 1:
      numItems = 5;
      break;

   case 3:
      numItems = 12;
      break;

   case 4:
      numItems = 99;
      break;

   default:
      numItems = 55;
      break;
}
```

#	Question	Your answer
1	userVal = 3;	
2	userVal = 0;	
3	userVal = 2;	

Construct 3.5.1: Switch statement general form.

```
switch (expression) {
   case constantExpr1:
      // Statements
      break;

   case constantExpr2:
      // Statements
      break;

   ...

   default: // If no other case matches
      // Statements
      break;
}
```

The switch statement's expression should be an integer or char. The expression should not be a string or a floating-point type. Each case must have a constant expression like 2 or 'q'; a case expression cannot be a variable.

Good practice is to always have a default case for a switch statement. A programmer may be sure all cases are covered only to be surprised that some case was missing.

P Participation Activity

3.5.3: Switch statement: Numbers to words.

Extend the program for dogYears to support age of 6 to 10 years. Conversions are 6:42, 7:47, 8:52, 9:57, 10:62.

Omitting the ***break*** statement for a case will cause the statements within the next case to be executed. Such "falling through" to the next case can be useful when multiple cases, such as cases 0, 1, and 2, should execute the same statements.

The following extends the previous program for dog ages less than 1 year old. If the dog's age is 0, the program asks for the dog's age in months. Within the `switch (dogAgeMonths)` statement, "falling through" is used to execute the same display statement for several values of dogAgeMonths. For example, if dogAgeMonths is 0, 1 or 2, the same the statement executes.

Figure 3.5.3: Switch example: Dog years with months.

```
#include <stdio.h>

int main(void) {
   int dogAgeYears  = 0;
   int dogAgeMonths = 0;

   printf("Enter your dog's age (in years): ");
   scanf("%d", &dogAgeYears);

   if (dogAgeYears == 0) {
      printf("Enter your dog's age in months: ");
      scanf("%d", &dogAgeMonths);

      switch (dogAgeMonths) {
         case 0:
         case 1:
         case 2:
            printf("That's 0..14 human months.\n");
            break;

         case 3:
         case 4:
         case 5:
         case 6:
            printf("That's 1..5 human years.\n");
            break;

         case 7:
         case 8:
            printf("That's 5..9 human years.\n");
            break;

         case 9:
         case 10:
         case 11:
         case 12:
            printf("That's 9..15 human years.\n");
            break;

         default:
            printf("Invalid input.\n");
            break;
      }
   }
   else {
      printf("FIXME: Do earlier dog year cases.\n");
      switch (dogAgeYears) {
      }
   }

   return 0;
}
```

```
Enter your dog's age (in ye
Enter your dog's age in mon
That's 5..9 human years.

...

Enter your dog's age (in ye
FIXME: Do earlier dog year
```

The order of cases doesn't matter assuming break statements exist at the end of each case. The earlier program could have been written with case 3 first, then case 2, then case 0, then case 1, for example (though that would be bad style).

A common error occurs when the programmer forgets to include a break statement at the end of a case's statements.

P Participation Activity

3.5.4: Switch statement.

userChar is a char and encodedVal is an int. What will encodedVal be for each userChar value?

```
switch (userChar) {
   case 'A':
      encodedVal = 1;
      break;

   case 'B':
      encodedVal = 2;
      break;

   case 'C':

   case 'D':
      encodedVal = 4;
      break;

   case 'E':
      encodedVal = 5;

   case 'F':
      encodedVal = 6;
      break;

   default:
      encodedVal = -1;
      break;
}
```

#	Question	Your answer
1	userChar = 'A'	
2	userChar = 'B'	
3	userChar = 'C'	
4	userChar = 'E'	
5	userChar = 'G'	

C Challenge Activity

3.5.1: Rock-paper-scissors.

C Challenge Activity

3.5.2: Switch statement to convert letters to Greek letters.

Section 3.6 - Boolean data types

Boolean refers to a quantity that has only two possible values, true or false.

The language has the built-in data type ***bool*** for representing Boolean quantities.

The programmer must add #include <stdbool.h> to use bool. The stdbool.h library was added to the C programming language in 1999, known as C99. When using an older version of C, such as C89, the bool data type is commonly defined using the following:

Figure 3.6.1: Defining bool data type.

```
typedef int bool;
#define false 0
#define true 1
```

The first statement defines a new type named bool that is equivalent to the int type. The next two statements define false as the value 0 and true as the value 1.

Figure 3.6.2: Example using variables of bool data type.

```
#include <stdio.h>
#include <stdbool.h>

int main(void) {
   bool isLarge = false;
   bool isNeg   = false;
   int  userNum = 0;

   printf("Enter any integer: ");
   scanf("%d", &userNum);

   if ((userNum < -100) || (userNum > 100)) {
      isLarge = true;
   }
   else {
      isLarge = false;
   }

   // Alternative way to set a bool var
   isNeg = (userNum < 0);

   printf("(isLarge: %d", isLarge);
   printf(" isNeg: %d)\n", isNeg);

   printf("You entered a ");
   if (isLarge && isNeg) {
      printf("large negative number.\n");
   }
   else if (isLarge && !isNeg) {
      printf("large positive number.\n");
   }
   else {
      printf("small number.\n");
   }

   return 0;
}
```

```
Enter any integer: 55
(isLarge: 0 isNeg: 0)
You entered a small number.

...

Enter any integer: -999
(isLarge: 1 isNeg: 1)
You entered a large negative numbe
```

A Boolean variable may be set using true or false keywords, as for `isLarge` above. Alternatively, a Boolean variable may be set to the result of a logical expression, which evaluates to true or false, as for `isNeg` above.

Unfortunately, the bool data type is not strictly Boolean. false is stored as 0, and true is stored as 1, as the above example shows for output values of `isLarge` and `isNeg`. Good practice is to avoid use of 0 or 1 values, e.g., avoiding a comparison like `isLarge == 0` or avoiding a computation like `numFeatures = isLarge + isNeg;`.

P Participation Activity

3.6.1: Boolean variables.

#	Question	Your answer
1	Write a statement to declare and initialize a Boolean variable named `night` to false.	

2	What is stored in variable `isFamous` after executing the following statements, `true` or `false`?
	`bool isTall = false;`
	`bool isRich = true;`
	`bool isFamous = false;`
	`if (isTall && isRich)`
	`{`
	`   isFamous = true;`
	`}`	

C Challenge Activity

3.6.1: Using bool.

C Challenge Activity

3.6.2: Bool in branching statements.

Section 3.7 - String comparisons

Two strings are commonly compared for equality. Equal strings have the same number of characters, and each corresponding character is identical.

P Participation Activity

3.7.1: Equal strings.

Which strings are equal?

#	Question	Your answer
1	"Apple", "Apple"	Equal
		Unequal
2	"Apple", "Apples"	Equal
		Unequal
3	"Apple pie!!", "Apple pie!!"	Equal
		Unequal
4	"Apple", "apple"	Equal
		Unequal

A programmer can compare two strings using: `strcmp(str1, str2) == 0`. The ***strcmp*** function returns 0 if the strings are equal, and some non-zero value otherwise. The programmer must add `#include <string.h>` to use strcmp. A common error is to omit the == 0 from strcmp(str1, str2) == 0 when comparing for equality. Another common error is to compare two strings using str1 == str2, which behaves differently than expected.

P Participation Activity

3.7.2: Comparing strings for equality.

To what does each expression evaluate? Assume str1 is "Apples" and str2 is "apples".

#	Question	Your answer
1	strcmp(str1, "Apples") == 0	True False
2	strcmp(str1, str2) == 0	True False
3	strcmp(str2, "oranges") != 0	True False
4	A good way to compare strings is: str1 == str2	True False
5	The following evaluates to true if the strings are equal: strcmp(str3, str4).	True False

Figure 3.7.1: String equality example: Censoring.

```c
#include <stdio.h>
#include <string.h>

int main() {
   char userWord[50] = "";

   printf("Enter a word: ");
   scanf("%s", userWord);

   if (strcmp(userWord, "Voldemort") == 0) {
      printf("He who must not be named\n");
   }
   else {
      printf("%s\n", userWord);
   }

   return 0;
}
```

```
Enter a word: Sally
Sally

...

Enter a word: Voldemort
He who must not be named

...

Enter a word: voldemort
voldemort
```

Strings are sometimes compared relationally (less-than, greater-than), as when sorting words alphabetically. For example, banana comes before orange alphabetically, so banana is less-than orange. Also, banana is less-than bananas.

To support relational comparisons, strcmp returns negative or positive values too, as follows.

Table 3.7.1: strcmp(str1, str2) return values.

Relation	Returns	Expression to detect
str1 less-than str2	Negative number	strcmp(str1, str2) < 0
str1 equal-to str2	0	strcmp(str1, str2) == 0
str1 greater-than str2	Positive number	strcmp(str1, str2) > 0

P Participation Activity

3.7.3: Relational string comparison.

#	Question	Your answer

1	Complete the code by comparing string variables myName and yourName. Make myName the first strcmp argument.	`if (  ) {`
		`  printf("%s is greater.", myName);`
		`}`

String comparisons treat uppercase and lowercase differently than most people expect. When comparing each character, the ASCII values are actually compared. 'A' is 65, B' is 66, etc., while 'a' is 97, 'b' is 98, etc. So "Apples" is less than "apples" or "abyss" because 'A' is less than 'a'. "Zoology" is less than "apples". A common error is to forget that case matters in a string comparison.

P Participation Activity

3.7.4: String comparison.

P Participation Activity

3.7.5: Case matters in string comparisons.

Indicate the result of comparing the first string with the second string.

#	Question	Your answer
1	"Apples", "Oranges"	less-than
		equal
		greater-than
2	"merry", "Merry"	less-than
		equal
		greater-than
3	"banana", "bananarama"	less-than
		equal
		greater-than

A programmer can compare strings while ignoring case by first converting both strings to lowercase before comparing (discussed elsewhere).

C Challenge Activity

3.7.1: String comparison: Detect word.

C Challenge Activity

3.7.2: Print two strings in alphabetical order.

Section 3.8 - String access operations

A string is a sequence of characters in memory. Each string character has a position number called an ***index***. The numbering starts with 0, not 1.

The notation someString[0] accesses the character at a particular index of a string, in this case index 0.

Figure 3.8.1: String character access.

```
#include <stdio.h>
#include <string.h>

int main() {
   char usrWord[20] = "";

   printf("Enter word (>= 5 letters): ");
   scanf("%s", usrWord);

   // Note: Error if usrWord has < 5 letters

   printf("Original: %s\n", usrWord);

   usrWord[0] = usrWord[3];
   usrWord[2] = usrWord[1];
   usrWord[1] = usrWord[4];

   printf("Scrambled: %s\n", usrWord);

   return 0;
}
```

```
Enter word (>= 5 letters): forest
Original: forest
Scrambled: esoest
```

P Participation Activity

3.8.1: String access.

Given userText is "Think".
Do not type quotes in your answers.

#	Question	Your answer
1	How many numbers do you see: 0 1 2 3	
2	What character is at index 1 of userText?	
3	What is the index of the last character, 'k', in userText?	
4	To what character does this evaluate: userText[3]	
5	What is userText after the following: userText[0] = 't';	

So that code can detect where a string ends, the compiler ends a string with a ***null character***, written as '\0'. The string's char array must be large enough to include the null character; "Hi" requires a char array of size at least 3 for the 'H', 'i', and the null character.

P Participation Activity

3.8.2: A char array definition and initialization with null-terminated string.

The string library provides useful functions for accessing information about a string. Most require knowledge of "pointers" so are introduced later. One function is introduced here.

Table 3.8.1: strlen() function.

strlen(someString)	Number of characters	`// char userText[50]`
		`// userText is "Help me!"`
		`strlen(userText)   // Returns 8`
		`// userText is ""`
		`strlen(userText)   // Returns 0`

Note that strlen is independent of the string size (50 above). Also, strlen does not count the null character that ends a string.

P Participation Activity

3.8.3: String length.

Given userText is "March 17, 2034".
Do not types quotes in answers.

#	Question	Your answer
1	What does strlen(userText) return?	
2	What is the index of the last character in userText?	
3	What character does userText[strlen(userText) - 1] return?	

A common error is to access an invalid array index, especially exactly one larger than the largest index. Given userText with size 8, the range of valid indices are 0..7; accessing with index 8 is an error.

P Participation Activity

3.8.4: String access.

If userText has size 5, reading userText[5] reads a memory location that may belong to another variable, thus yielding a strange value. Likewise, assigning a value to userText[5] may overwrite the value in some other variable, yielding bizarre program behavior. Such an error can be extremely difficult to debug.

P Participation Activity

3.8.5: Out-of-range string access.

Given userText = "Monday".

#	Question	Your answer
1	userText[7] = '!' may write to another variable's location and cause bizarre program behavior.	True False
2	userText[strlen(userText)] yields 'y'.	True False

C Challenge Activity

3.8.1: String library functions.

Section 3.9 - Character operations

Including the ***ctype.h library*** via via `#include <ctype.h>` provides access to several functions for working with characters. ctype stands for character type.

Table 3.9.1: Character functions return values.

isalpha(c)	true if alphabetic: a-z or A-Z	`isalpha('x') // true`
		`isalpha('6') // false`
		`isalpha('!') // false`
				toupper(c)	Uppercase version	`toupper`
						`toupper`
						`toupper`

isdigit(c)	true if digit: 0-9.	`isdigit('x') // false`
		`isdigit('6') // true`
				tolower(c)	Lowercase version	`tolower`
						`tolower`
						`tolower`

isspace(c)	true if whitespace.	`isspace(' ')  // true`
		`isspace('\n') // true`
		`isspace('x')  // false`				

Note: Above, false is zero, and true is non-zero.

See http://www.cplusplus.com/reference/cctype/ for a more complete list (applies to both C and C++).

P Participation Activity

3.9.1: Character functions.

To what value does each evaluate? userStr is "Hey #1?".

#	Question	Your answer
1	isalpha('7')	True False
	isalpha(userStr[0])	True

2	isalpha(userStr[0])	True False
3	isspace(userStr[3])	True False
4	isdigit(userStr[6])	True False
5	toupper(userStr[1]) returns 'E'.	True False
6	tolower(userStr[2]) yields an error because 'y' is already lower case .	True False
7	tolower(userStr[6]) yields an error because '?' is not alphabetic.	True False
8	After tolower(userStr[0]), userStr becomes "hey #1?"	True False

C Challenge Activity

3.9.1: String with digit.

C Challenge Activity

3.9.2: Whitespace replace.

Section 3.10 - Conditional expressions

If-else statements with the form shown below are so common that the language supports the shorthand notation shown.

P Participation Activity

3.10.1: Conditional expression.

A ***conditional expression*** has the following form:

Construct 3.10.1: Conditional expression.

```
condition ? exprWhenTrue : exprWhenFalse
```

All three operands are expressions. If the `condition` evaluates to true, then `exprWhenTrue` is evaluated. If the condition evaluates to false, then `exprWhenFalse` is evaluated. The conditional expression evaluates to whichever of those two expressions was evaluated. For example, if x is 2, then the conditional expression `(x == 2) ? 5 : 9 * x` evaluates to 5.

A conditional expression has three operands and thus the "?" and ":" together are sometimes referred to as a ***ternary operator***.

Good practice is to restrict usage of conditional expressions to an assignment statement, as in: y = (x == 2) ? 5 : 9 * x;. Common practice is to put parentheses around the first expression of the conditional expression, to enhance readability.

P Participation Activity

3.10.2: Conditional expressions.

Convert each if-else statement to a single assignment statement using a conditional expression, using parentheses around the condition. Enter "Not possible" if appropriate. ..

#	Question	Your answer

1	`if (x > 50) {`
	`   y = 50;`
	`}`
	`else {`
	`   y = x;`
	`}`
		`y = (` ☐ `) ? 50 : x;`

2	`if (x < 20) {`
	`   y = x;`
	`}`
	`else {`
	`   y = 20;`
	`}`
		`y = (x < 20)` ☐

3	`if (x < 100) {`
	`   y = 0;`
	`}`
	`else {`
	`   y = x;`
	`}`
		☐

4	`if (x < 0) {`
	`   x = -x;`
	`}`
	`else {`
	`   x = x;`
	`}`
		☐

5	`if (x < 0) {`
	`   y = -x;`
	`}`
	`else {`
	`   z = x;`
	`}`
		☐

C Challenge Activity

3.10.1: Conditional expression: Print negative or positive.

C Challenge Activity

3.10.2: Conditional assignment

Section 3.11 - Floating-point comparison

Floating-point numbers should not be compared using ==. Ex: Avoid float1 == float2. Reason: Some floating-point numbers cannot be exactly represented in the limited available memory bits like 64 bits. Floating-point numbers expected to be equal may be close but not exactly equal.

P Participation Activity

3.11.1: Floating-point comparisons.

Floating-point numbers should be compared for "close enough" rather than exact equality. Ex: If (x - y) < 0.0001, x and y are deemed equal. Because the difference may be negative, the absolute value is used: fabs(x - y) < 0.0001. fabs() is a function in the math library. The difference threshold indicating that floating-point numbers are equal is often called the ***epsilon***. Epsilon's value depends on the program's expected values, but 0.0001 is common.

P Participation Activity

3.11.2: Using == with floating-point numbers.

#	Question	Your answer
1	Given: float x, y x == y is OK.	True False
2	Given: double x, y x == y is OK.	True False
3	Given: double x x == 32.0 is OK.	True False
4	Given: int x, y x == y is OK.	True False
5	Given: double x x == 32 is OK.	True False

P Participation Activity

3.11.3: Floating-point comparisons.

Each comparison has a problem. Click on the problem.

#	Question
1	fabs (x - y) == 0.0001
2	abs (x - y) < 0.0001
3	fabs (x - y) < 1.0

P Participation Activity

3.11.4: Floating point statements.

Complete the comparison for floating-point numbers.

#	Question	Your answer
1	Determine if double variable x is 98.6.	`□ (x - 98.6) < 0.0001`
2	Determine if double variables x and y are equal. Threshold is 0.0001.	`fabs(x - y) □`
3	Determine if double variable x is 1.0	`fabs( □ ) < 0.0001`

Figure 3.11.1: Example of comparing floating-point numbers for equality: Body temperature.

```
#include <stdio.h>
#include <math.h>

int main(void) {
   double bodyTemp = 0.0;

   printf("Enter body temperature in Fahrenheit: ");
   scanf("%lf", &bodyTemp);

   if (fabs(bodyTemp - 98.6) < 0.0001) {
      printf("Temperature is exactly normal.\n");
   }
   else if (bodyTemp > 98.6) {
      printf("Temperature is above normal.\n");
   }
   else {
      printf("Temperature is below normal.\n");
   }

   return 0;
}
```

```
Enter body temperature in F
Temperature is exactly norm

Enter body temperature in F
Temperature is below normal

Enter body temperature in F
Temperature is above normal
```

Participation Activity

3.11.5: Body temperature in Fahrenheit.

Refer to the body temperature code provided in the previous figure.

#	Question	Your answer
1	What is output if the user enters 98.6?	Exactly normal
		Above normal
		Below normal
2	What is output if the user enters 97.0?	Exactly normal
		Above normal
		Below normal
3	What is output if the user enters 98.6000001?	Exactly normal
		Above normal
		Below normal

To see the inexact value stored in a floating-point variable, a format sub-specifier can be used in an output statement. Such output formatting is discussed in another section.

Figure 3.11.2: Observing the inexact values stored in floating-point variables.

```
#include <stdio.h>

int main(void) {
   double sampleValue1 = 0.2;
   double sampleValue2 = 0.3;
   double sampleValue3 = 0.7;
   double sampleValue4 = 0.0;
   double sampleValue5 = 0.25;

   printf("samplevalue1 using just %%lf: %lf\n",
          sampleValue1);

   printf("sampleValue1 is %.25lf\n", sampleValue1);
   printf("sampleValue2 is %.25lf\n", sampleValue2);
   printf("sampleValue3 is %.25lf\n", sampleValue3);
   printf("sampleValue4 is %.25lf\n", sampleValue4);
   printf("sampleValue5 is %.25lf\n", sampleValue5);

   return 0;
}
```

```
samplevalue1 using
sampleValue1 is 0.2
sampleValue2 is 0.2
sampleValue3 is 0.6
sampleValue4 is 0.0
sampleValue5 is 0.2
```

P Participation Activity

3.11.6: Inexact representation of floating-point values.

P Participation Activity

3.11.7: Representing floating-point numbers.

#	Question	Your answer
1	Floating-point values are always stored with some inaccuracy.	True False
2	If a floating-point variable is assigned with 0.2, and prints as 0.2, the value must have been represented exactly.	True False

C Challenge Activity

3.11.1: Floating-point comparison: Print Equal or Not equal.

Section 3.12 - Short circuit evaluation

A logical operator evaluates operands from left to right. ***Short circuit evaluation*** skips evaluating later operands if the result of the logical operator can already be determined. The logical AND operator short circuits to false if the first operand evaluates to false, and skips evaluating the second operand. The logical OR operator short circuits to true if the first operand is true, and skips evaluating the second operand.

P Participation Activity

3.12.1: Short circuit evaluation: Logical AND.

Table 3.12.1: Short circuit evaluation.

Operator	Example	Short circuit evaluation
`operand1 && operand2`	**`true`** `&& operand2`	If the first operand evaluates to true, operand2 is evaluated.
	`false` `&& operand2`	If the first operand evaluates to false, the result of the AND operation is always false, so operand2 is not evaluated.
`operand1 || operand2`	**`true`** `|| operand2`	If the first operand evaluates to true, the result of the OR operation is always true, so operand2 is not evaluated.
	`false` `|| operand2`	If the first operand evaluates to false, operand2 is evaluated.

P Participation Activity

3.12.2: Determine which operands the program evaluates.

#	Question	Your answer
1	`(x < 4) && (y > 3)` What value of x results in short circuit evaluation, which skips evaluating the second operand?	6 2 3
2	`(y == 3) || (x > 2)` What value of y results in short circuit evaluation, which skips evaluating the second operand?	2 4 3
3	`(y < 3) || (x == 1)` What value of y does not result in short circuit evaluation, such that both operands are evaluated?	3 1 2

4	`(x < 3) && (y < 2) && (z == 5)`
	What values of x and y do not result in short circuit evaluation, such that all operands are evaluated?
		x = 2, y = 2
		x = 1, y = 0
		x = 4, y = 1
		x = 3, y = 2

5	`((x > 2) || (y < 4)) && (z == 10)` Given x = 4, y = 1, and z = 10, which comparisons are evaluated?	(x > 2), (y < 4), and (z == 10) (x > 2) and (z == 10) (x > 2) and (y < 4)

Section 3.13 - C example: Salary calculation with branches

P Participation Activity

3.13.1: Calculate salary: Calculate overtime using branches.

The following program calculates yearly and monthly salary given an hourly wage. The program assumes work-hours-per-week limit of 40 and work-weeks-per-year of 50.

Overtime refers to hours worked per week in excess of some weekly limit, such as 40 hours. Some companies pay time-and-a-half for overtime hours, meaning overtime hours are paid at 1.5 times the hourly wage.

Overtime pay can be calculated with pseudocode as follows (assuming a weekly limit of 40 hours):

```
weeklyLimit = 40
if weeklyHours <= weeklyLimit
   weeklySalary = hourlyWage * weeklyHours
else
   overtimeHours = weeklyHours - weeklyLimit
   weeklySalary = hourlyWage * weeklyLimit + (overtimeHours *
hourlyWage * 1.5)
```

1. Run the program and observe the salary earned.
2. Modify the program to read user input for weeklySalary. Run the program again.

P Participation Activity

3.13.2: Determine tax rate.

Income tax is calculated based on annual income. The tax rate is determined with a tiered approach: Income above a particular tier level is taxed at that level's rate.

1. Run the program with an annual income of 120000. Note the tax rate and tax to pay.
2. Modify the program to add a new tier: Annual income above 50000 but less than or equal to 100000 is taxed at the rate of 30%, and annual income above 100000 is taxed at 40%.
3. Run the program again with an annual income of 120000. What is the tax rate and tax to pay now?
4. Run the program again with an annual income of 60000. (Change the input area below the program.)
5. Challenge: What happens if a negative annual salary is entered? Modify the program to print an error message in that case.

Section 3.14 - C example: Search for name using branches

P Participation Activity

3.14.1: Search for name using branches.

A ***core generic top-level domain (core gTLD)*** name is one of the following Internet domains: .com, .net, .org, and .info (Wikipedia: gTLDs). The following program asks the user to input a name and prints whether that name is a gTLD. The program uses the strcmp() function, which returns zero if the two compared strings are identical.

1. Run the program, noting that the .info input name is not currently recognized as a gTLD.
2. Extend the if-else statement to detect the .info domain name as a gTLD. Run the program again.
3. Extend the program to allow the user to enter the name with or without the leading dot, so .com or just com.

Below is a solution to the above problem.

P Participation Activity

3.14.2: Search for name using branches (solution).

Section 3.15 - Warm up: Automobile service cost (C)

(1) Prompt the user for an automobile service. Each service type is composed of two strings. Output the user's input. (1 pt)

Ex:

```
Enter the desired auto service: Oil change
You entered: Oil change
```

(2) Output the price of the requested service. (4 pts)

Ex:

```
Cost of oil change: $35
```

The program should support the following services:

- Oil change -- $35
- Tire rotation -- $19
- Car wash -- $7

If the user enters a service that is not listed above, then output the following error message:

```
Error: Requested service is not recognized
```

L Lab Submission

3.15.1: Warm up: Automobile service cost (C)

File: main.c

```
Loading latest submission...
```

Develop | Submit

In "Develop" mode, you can run your program a
values (if desired) in the first box below, then clic
second box.

RUN PROGRAM

Enter program input (optional)

(Optional)

main.c (Your progra

Program output displayed here

Section 3.16 - Program: Automobile service invoice (C)

(1) Output a menu of automotive services and the corresponding cost of each service. (2 pts)

Ex:

```
Davy's auto shop services
Oil change -- $35
Tire rotation -- $19
Car wash -- $7
Car wax -- $12
```

(2) Prompt the user for two services. Each service type is composed of two strings. (2 pts)

Ex:

```
Select first service: Oil change

Select second service: Car wax
```

(3) Output an invoice for the services selected. Output the cost for each service and the total cost. (3 pts)

```
Davy's auto shop invoice

Service 1: Oil change, $35
Service 2: Car wax, $12

Total: $47
```

(4) Extend the program to allow the user to enter a dash (-), which indicates no service. (3 pts)

Ex:

```
Select first service: Tire rotation

Select second service: -
```

```
Davy's auto shop invoice

Service 1: Tire rotation, $19
Service 2: No service

Total: $19
```

Lab
Submission

3.16.1: Program: Automobile service invoice (C)

```
File: main.c
1 Loading latest submission...
```

Develop | Submit

In "Develop" mode, you can run your program a
values (if desired) in the first box below, then clic
second box.

RUN PROGRAM

Enter program input (optional)

(Optional)

↓

main.c (Your progra

↓

Program output displayed here