3.1 If-else branches (general)

Branch concept

People familiar with restaurants may be familiar with steering people to different-sized tables based on group size.

PARTICIPATION ACTIVITY 3.1.1: Branching concept.

Animation captions:

1. A restaurant host seats patrons. The host seats a party of 1 at the counter.
2. A party of 2 is seated at a small table. Other size parties are seated at a large table.
3. The host mentally executes the algorithm: If party of 1, seat at counter; Else If party of 2, seat at small table; Else seat at large table.

PARTICIPATION ACTIVITY 3.1.2: Branch concept.

Consider the example above.

1) A party of 1 is sat at _____ .
   - ○ the counter
   - ○ a small table

2) A party of 2 is sat at _____ .
   - ○ the counter
   - ○ a small table

3) A party of 5 is sat _____ .
   - ○ at a large table
   - ○ nowhere

Branch basics (If)

A ***branch*** is a program path taken only if an expression's value is true. Ex: A hotel may discount a price only for people over age 60.

PARTICIPATION ACTIVITY 3.1.3: A simple branch: Hotel discount.

Animation captions:

1. A branch is a program path taken if an expression is true. This program assigns price = 155, and gets input age. If age > 60, the branch with price = price - 20 should be taken.
2. When executing, if the input age is 72, then age > 60 is true. The branch is taken, so price = price - 20 executes. The output is thus 135.
3. But, if the input age were 45, then age > 60 is false. The branch is not taken, so the output is 155.

Such a branch is commonly known as an ***If*** branch: A branch taken only if an expression is true.

PARTICIPATION ACTIVITY 3.1.4: Branches.

Consider the example above.

1) If the input is 25, the output is Cost: _____ .
   - ○ 135
   - ○ 155

2) If the input is 80, the output is Cost: _____ .
   - ○ 135
   - ○ 155

3) If the input is 60, will the branch be taken?
   - ○ Yes
   - ○ No

If branch example: Absolute value

The following shows how an if branch can be used to compute an absolute value of a number.

PARTICIPATION ACTIVITY 3.1.5: Example if branch: Computing absolute value.

Animation captions:

1. This program outputs the absolute value of the input. After getting input val, if val < 0, the branch val = -val executes. Finally, val is output.
2. When executing, if the input val is -9, then val < 0 is true. The branch is taken, so val = -val executes, assigning val with 9. So 9 is output.
3. But, if the input val were 45, then val < 0 is false. The branch is not taken, so the output is 45.

PARTICIPATION ACTIVITY 3.1.6: Example if branch: Absolute value.

Consider the example above.

1) If the input is -6, does the branch execute?
   - Yes
   - No

2) If the input is 0, does the branch execute?
   - Yes
   - No

If-else branches

An ***if-else*** structure has two branches: The first branch is taken if an expression is true, else the other branch is taken.

PARTICIPATION ACTIVITY 3.1.7: If-else branches.

Animation captions:

1. An if-else has two branches. The if branch is taken if the expression is true. Otherwise, the else branch is taken.
2. This program should output "Can ride. Done." if the input weight < 150, else should output "Can't ride. Done."
3. If the input weight is 128, weight < 150 is true, so the if branch executes. "Can ride" is output.
4. But, if the input is 175, weight < 150 is false, so the else branch executes. "Can't ride" is output.

PARTICIPATION ACTIVITY 3.1.8: If-else branches.

Consider the example above.

1) What is output when the input is 90?

- Can ride. Done.
- Can't ride. Done.

2) What is output when the input is 200?

- Can ride. Done.
- Can't ride. Done.

3) What is output when the input is exactly 150?

- Can ride. Done. Can't ride. Done.
- Can ride. Can't ride. Done.
- Can't ride. Done.

4) What input value causes both the if branch to execute (outputting "Can ride") and the else branch to execute (outputting "Can't ride")?

- 149
- 150
- 151
- No such value

5) What value causes "Done." to NOT be output?

- 130
- 160
- No such value.

If-else example: Max

The following example shows how an if-else can be used to get the maximum of two values.

PARTICIPATION ACTIVITY 3.1.9: If-else example: Max.

Animation captions:

1. This program should output the maximum of two input values.
2. If x is 7 and y is 3, x > y is true, so the if branch executes, which assigns max with x, so 7.

3. If the input instead is 3 7, x > y is false, so the else executes. max is assigned with y, which is 7. Thus, whether the input is 7 3 or is 3 7, max gets the larger value.

PARTICIPATION ACTIVITY 3.1.10: If-else example: Max.

Consider the example above.

1) When the input is -3 0, which branch executes?
   - If
   - Else

2) When the input is 99 98, which branch executes?
   - If
   - Else

3) The if branch assigns max = x. The else branch assigns max = ?
   - x
   - y

4) If the inputs are 5 5, does max get assigned with x or y?
   - x
   - y

If-elseif-else branches

Commonly a programmer wishes to take one of multiple (three or more) branches. An if-else can be extended to an if-elseif-else structure. Each branch's expression is checked in sequence; as soon as one branch's expression is found to be true, that branch is taken. If no expression is found true, execution will reach the else branch, which then executes.

PARTICIPATION ACTIVITY 3.1.11: If-elseif-else branch.

Animation captions:

1. This program gets a year, then if 1984 outputs "Orwell!", else if 2001 outputs "Space Odyssey!", else outputs "Nothing special", thus indicating if a year was featured in a famous book/movie.

2. If year equals 1984, the if branch is taken, outputting "Orwell!".
3. If the input is 2001, year equals 1984 is false. So the next branch is checked: year equals 2001 is true, so that branch is taken, thus outputting "Space Odyssey!".
4. If year equals anything else, like 1997, the if branch is not taken, and else-if branch is not taken, so the else branch is taken. "Nothing special" is output.

Note: The else part is optional. If omitted, then if none of the previous expressions are true, no branch executes.

PARTICIPATION ACTIVITY 3.1.12: If-elseif-else.

Consider the if-elseif-else structure below:

```
If x equals -1
     Put "Disagrees" to output

Else If x equals 0
     Put "Neutral" to output

Else If x equals 1
     Put "Agrees" to output

Else
     Put "Invalid entry" to output
```

1) If x is 1, what is output?

- Disagrees
- Neutral
- Agrees
- Invalid entry

2) If x is -2, what is output?

- Disagrees
- Invalid entry
- (Nothing is output)

3) Could the programmer have written the three branches in the order x equals 1, x equals 0, and x equals -1, and achieved the same results?

- No
- Yes

4) In the code above, suppose a programmer, after the third branch (x

equals 1), inserts a new branch: Else If x equals -1 ... When might that new branch execute?

- When x is -1
- When x is 1
- Never

5) In the code above, suppose a programmer removed the Else part entirely. If x is 2, which is correct?

- The last branch, meaning the Else If x equals 1 branch, will execute.
- No branch will execute.
- The program is not legal.

3.2 If-else

If statements

An ***if*** statement executes a group of statements if an expression is true. Braces surround the if branch's statements. ***Braces*** { }, sometimes redundantly called curly braces, represent a grouping, such as a grouping of statements. Note: { } are braces, [ ] are brackets.

Good practice is to indent a branch's statements, using a consistent number of spaces. This material indents 3 spaces.

PARTICIPATION ACTIVITY 3.2.1: if statement: Hotel discount.

Animation captions:

1. An if statement executes a group of statements if an expression is true. The program assigns hotelRate with 155 and then gets the user's age from input.
2. userAge is 68, so the expression 68 > 65 is true, and the if's statement will execute. Thus, the statement following the opening brace { will execute next.
3. hotelRate is decreased by 20, which is the discount for guests older than 65.
4. The closing brace } indicates the end of the group of statements.
5. The program completes by printing the hotel rate.

PARTICIPATION ACTIVITY 3.2.2: If statement.

What is the final value of numItems?

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


Check Show answer

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

Check Show answer

If-else statement

An ***if-else*** statement executes one group of statements when an expression is true, and another group of statements when the expression is false.

Construct 3.2.1: If-else statement.

```
// Statements that execute before the branches

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

// Statements that execute after the branches
```


In the example below, if a user inputs an age less than 25, the statement `insurancePrice = 4800` executes. Otherwise, `insurancePrice = 2200` executes.

PARTICIPATION

ACTIVITY 3.2.3: if-else statement: Car insurance.

Animation captions:

1. An if-else statement executes a group of statements if an expression is true, and executes another group of statements otherwise.
2. userAge is 22, so the expression 22 < 25 is true, and the if's statements will execute.
3. user's age is 40, so the expression 40 < 25 is false, and the else's statements will execute.

Car insurance prices

(Car insurance prices for drivers 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).

PARTICIPATION ACTIVITY 3.2.4: If-else statements.

1) What is the final value of numItems?

```
bonusVal = 5;
if (bonusVal < 12) {
   numItems = 100;
}
else {
   numItems = 200;
}
```

Check Show answer

2) What is the final value of numItems?

```
bonusVal = 12;
if (bonusVal < 12) {
   numItems = 100;
}
else {
   numItems = 200;
}
```


Check Show answer

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;
```


Check Show answer

4) What is the final value of bonusVal?

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

Check Show answer

5) What is the final value of bonusVal?

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

Check Show answer

PARTICIPATION ACTIVITY 3.2.5: 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).

1) If userAge is greater than 62, assign itemDiscount with 15. Else, assign itemDiscount with 0.

Check Show answer

2) If numPeople is greater than 10, execute groupSize = 2 * groupSize. Otherwise, execute groupSize = 3 * groupSize and numPeople = numPeople - 1.

Check Show answer

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.

Check Show answer

CHALLENGE ACTIVITY 3.2.1: Enter the output for the if-else branches.

Start

Type the program's output

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

int main() {
   int numApples;

   numApples = 4;

   if (numApples < 3) {
      cout << "c" << endl;
   }
   else {
      cout << "f" << endl;
   }

   cout << "k" << endl;

   return 0;
}
```


1 2

Check Next

CHALLENGE ACTIVITY 3.2.2: Basic if-else expression.

Start

Write an expression that will cause the following code to print "less than 20" if the value of userAge is less than 20.

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     int userAge;
6
7     cin >> userAge; // Program will be tested with values: 18, 19, 20, 21.
8
9     if (/* Your code goes here */) {
10       cout << "less than 20" << endl;
11    }
12    else {
13       cout << "20 or more" << endl;
14    }
15
16    return 0;
17 }
```


1 2 3

Check Next

CHALLENGE ACTIVITY 3.2.3: Basic if-else.

Start

Write an if-else statement for the following:

If userTickets is less than 8, execute awardPoints = 1. Else, execute awardPoints = userTickets.

Ex: If userTickets is 3, then awardPoints = 1.

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

int main() {
   int awardPoints;
   int userTickets;

   cin >> userTickets; // Program will be tested with values: 5, 6, 7, 8.

   /* Your code goes here */

   cout << awardPoints << endl;

   return 0;
}

```

1 2

Check Next

Multi-branch if-else statements

An If-else statement can be extended to have three (or more) branches. Each branch's expression is checked in sequence. As soon as one branch's expression is found to be true, that branch's statement execute (and no subsequent branch is considered). If no expression is true, the else branch executes.

Construct 3.2.2: Multi-branch if-else statement. Only 1 branch will execute.

```
if (expression1) {
   // Statements that execute when expression1 is true
   // (first branch)
}
else if (expression2) {
   // Statements that execute when expression1 is false and expression2 is true
   // (second branch)
}
else {
   // Statements that execute when expression1 is false and expression2 is false
   // (third branch)
}
```


The ***equality operator ==*** evaluates to true if the left side and right side are equal. Ex: If numYears holds the value 10, then the expression `numYears == 10` evaluates to true.

Note that the equality operator is ==, not =.

Figure 3.2.1: Multi-branch if-else example: Anniversaries.

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

int main() {
   int numYears;

   cout << "Enter number years married: ";
   cin  >> numYears;

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

   return 0;
}
```

```
Enter number years married: 10
A whole decade -- impressive.

...

Enter number years married: 25
Your silver anniversary --
enjoy.

...

Enter number years married: 30
Nothing special.

...

Enter number years married: 1
Your first year -- great!
```


PARTICIPATION ACTIVITY 3.2.6: Multi-branch if-else statements.

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;
}
```


1) numSales is 2

Check Show answer

2) numSales is 0

Check Show answer

3) numSales is 7

Check Show answer

Common errors

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.

PARTICIPATION ACTIVITY 3.2.7: Common error when omitting braces.

Animation captions:

1. Braces aren't used, so the else branch's only statement is totBonus = totBonus + 1. But, salesBonus = 20; should also be part of the else branch.
2. Always using braces avoids the common error of not including all statements within an if or else branch.

PARTICIPATION ACTIVITY 3.2.8: Braces are important.

Omitting braces is a common source of errors. What is the final value of numItems?

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


Check Show answer

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

Check Show answer

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

Check Show answer

CHALLENGE ACTIVITY 3.2.4: If-else statement: Fix errors.

Re-type the code and fix any errors. The code should convert non-positive numbers to 1.

```
if (userNum > 0)
   cout << "Positive." << endl;
else
   cout << "Not positive, converting to 1." << endl;
   userNum = 1;

cout << "Final: " << userNum << endl;
```



```
1 #include <iostream>
2 using namespace std;
```

```
3
4  int main() {
5     int userNum;
6
7     cin >> userNum;
8
9     /* Your solution goes here  */
10
11    return 0;
12 }
```


Run

View your last submission

3.3 More if-else

Nested if-else statements

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

Figure 3.3.1: Nested if-else.

```
if (userChoice == 1) { // userChoice is 1
   ...
}
else if (userChoice == 2) {
   if (numItems < 0) { // userChoice is 2 and numItems < 0
      ...
   }
   else {              // userChoice is 2 and numItems >= 0
      ...
   }
}
else {                 // userChoice is not 1 or 2
   ...
}
```

PARTICIPATION ACTIVITY 3.3.1: Nested if-else statements.

Determine the final value of salesBonus given the initial values specified below.

```
if (salesType == 2) {
   if (salesBonus < 5) {
     salesBonus = 10;
   }
   else {
      salesBonus = salesBonus + 2;
   }
}
else {
   salesBonus = salesBonus + 1;
}
```


1) salesType = 1; salesBonus = 0;

- 0
- 1
- 10

2) salesType = 2; salesBonus = 4;

- 5
- 6
- 10

3) salesType = 2; salesBonus = 7;

- 8
- 9
- 10

Multiple distinct if statements

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 3.3.2: Multiple distinct if statements.

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

int main() {
   int userAge;

   cout << "Enter age: ";
   cin  >> userAge;

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

   if (userAge > 15) {
      cout << "You are old enough to drive." << endl;
   }

   if (userAge > 17) {
      cout << "You are old enough to vote." << endl;
   }

   if (userAge > 24) {
      cout << "Most car rental companies will rent to
you." << endl;
   }

   if (userAge > 34) {
      cout << "You can run for president." << endl;
   }

   return 0;
}
```

```
Enter age: 12
Enjoy your early years.

...

Enter age: 27
You are old enough to drive.
You are old enough to vote.
Most car rental companies will
rent to you.
...
Enter age: 99
You are old enough to drive.
You are old enough to vote.
Most car rental companies will
rent to you.
You can run for president.
```


PARTICIPATION ACTIVITY 3.3.2: If statements.

Determine the final value of numBoxes.

1)

```
numBoxes  = 0;
numApples = 9;

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


Check Show answer

2)

```
numBoxes  = 0;
numApples = 9;

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


Check Show answer

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

Start

Type the program's output

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

int main() {
   int numItems;

   numItems = 6;

   if (numItems > 1) {
      cout << "a" << endl;
   }
   else if (numItems < 8) {
      cout << "d" << endl;
   }
   else {
      cout << "h" << endl;
   }

   cout << "p" << endl;

   return 0;
}
```


1 2 3

Check Next

CHALLENGE ACTIVITY 3.3.2: If-else statements.

Start

Write multiple if statements:
If carYear is before 1968, print "Probably has few safety features." (without quotes).
If after 1971, print "Probably has seat belts.".
If after 1990, print "Probably has electronic stability control.".
If after 2002, print "Probably has airbags.".
End each phrase with period and newline. Ex: carYear = 1995 prints:


```
Probably has seat belts.
Probably has electronic stability control.
```

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

int main() {
   int carYear;

   cin >> carYear;

   /* Your code goes here */

   return 0;
}
```

1 2

Check Next

3.4 Equality and relational operators

Equality operators

An ***equality operator*** checks whether two operands' values are the same (==) or different (!=). Note that equality is ==, not just =.

An expression involving an equality operator evaluates to a Boolean value. A ***Boolean*** is a type that has just two values: true or false.

Table 3.4.1: Equality operators.

Equality operators	Description	Example (assume x is 3)
==	a **==** b means a is equal to b	x == 3 is true x == 4 is false
!=	a **!=** b means a is not equal to b	x != 3 is false x != 4 is true

PARTICIPATION ACTIVITY 3.4.1: Evaluating expressions that have equality operators.

Indicate whether the expression evaluates to true or false.
x is 5, y is 7.

1) x == 5
   - True
   - False

2) x == y
   - True
   - False

3) y != 7
   - True
   - False

4) y != 99
   - True
   - False

5) x != y
   - True
   - False

6) Is x = = y a valid expression?

- Yes
- No

PARTICIPATION ACTIVITY 3.4.2: Creating expressions with equality operators.

Type the equality operator to complete the desired expression.

1) numDogs is 0

```
numDogs [ ] 0
```

Check Show answer

2) numDogs and numCats are the same

```
numDogs [ ] numCats
```

Check Show answer

3) numDogs and numCats differ

```
numDogs [ ] numCats
```

Check Show answer

4) numDogs is either less than or greater than numCats

```
numDogs [ ] numCats
```

Check Show answer

5) userChar is the character 'x'.

```
userChar [ ] 'x'
```

Check Show answer

Relational operators

A ***relational operator*** checks how one operand's value relates to another, like being greater than.

Some operators like >= involve two characters. A programmer cannot arbitrarily combine the >, =, and < symbols; only the shown two-character sequences represent valid operators.

Table 3.4.2: Relational operators.

Relational operators	Description	Example (assume x is 3)
<	a **<** b means a is less than b	x < 4 is true x < 3 is false
>	a **>** b means a is greater than b	x > 2 is true x > 3 is false
<=	a **<=** b means a is less than or equal to b	x <= 4 is true x <= 3 is true x <= 2 is false
>=	a **>=** b means a is greater than or equal to b	x >= 2 is true x >= 3 is true x >= 4 is false

PARTICIPATION ACTIVITY 3.4.3: Evaluating equations having relational operators.

Indicate whether the expression evaluates to true or false.
x is 5, y is 7.

1) x <= 7

- True
- False

2) y >= 7

- True
- False

3) Is x <> y a valid expression?

- Yes
- No

4) Is x =< y a valid expression?

- Yes
- No

PARTICIPATION ACTIVITY 3.4.4: Creating expressions with relational operators.

Type the operator to complete the desired expression.

1) numDogs is greater than 10

```
numDogs  10
```

Check Show answer

2) numCars is greater than or equal to 5

```
numCars  5
```

Check Show answer

3) numCars is 5 or greater

```
numCars  5
```

Check Show answer

4) centsLost is a negative number

```
centsLost  0
```

Check Show answer

Example: Golf scores

In golf and miniature golf, each hole has a "par" indicating the normal number of strokes to get the ball in the hole. The following program outputs special names for numbers below par. The program uses one relational operator, and several equality operators, in a multi-branch if-else statement.

zyDE 3.4.1: Golf scores.

This program outputs special names for different numbers of strokes for a par 4 hole. P
Run to see the name for the input strokes of 2. Try changing the input strokes to differer
values, and press Run again to see the name.

If desired, try extending the program to print "Bogey" for 5, and "Double bogey" for 6. Remember that equals is ==, not =.

Try typing "= 1" instead of "== 1" and see what happens. Try typing an unsupported oper like =>, and see what happens.

Load default template...

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

int main() {
   int numStrokes;

   cin >> numStrokes;

   // Assumes "par 4"
   if (numStrokes <= 0) {
      cout << "Invalid entry." << endl;
   }
   else if (numStrokes == 1) {
      cout << "Hole in 1!!!" << endl;
   }
   else if (numStrokes == 2) {
      cout << "Eagle!" << endl;
```

2

Run

CHALLENGE ACTIVITY 3.4.1: Enter the output for the branches with relational and equality operators.

Start

Type the program's output

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

int main() {
   int numEggs;

   numEggs = 6;

   if (numEggs <= 5) {
      cout << "b" << endl;
   }
   else {
      cout << "e" << endl;
   }

   cout << "g" << endl;

   return 0;
}
```

1 2 3

Check Next

CHALLENGE ACTIVITY 3.4.2: Equality and relational expressions.

Start

Write an expression that will cause "Foot or more" to print if the value of numInches is greater than or equal to 12.

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

int main() {
   int numInches = 0;

   cin >> numInches; // Program will be tested with values: 11, 12, 13, 14.

   if (/* Your code goes here */) {
      cout << "Foot or more" << endl;
   }
   else {
      cout << "Less than a foot" << endl;
   }

   return 0;
}
```

1 2 3

Check Next

Comparing characters, strings, and floating-point types

The relational and equality operators work for integer, character, and floating-point built-in types.

Comparing characters compares their ASCII numerical encoding.

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 can also be used for the string type. Strings are equal if they have the same number of characters and corresponding characters are identical. If string myStr = "Tuesday", then (myStr ==

"Tuesday") is true, while (myStr == "tuesday") is false because T differs from t.

PARTICIPATION ACTIVITY 3.4.5: Comparing various types.

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

1) myInt == 42
   - OK
   - Not OK

2) myChar == 'q'
   - OK
   - Not OK

3) myDouble == 3.26
   - OK
   - Not OK

4) myString == "Hello"
   - OK
   - Not OK

Common errors

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 code is not a syntax error. The statement assigns numDogs with 9, 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.

Another common error is to use invalid character sequences like =>, !<, or <>, which are *not* valid operators.

PARTICIPATION ACTIVITY 3.4.6: Watch out for assignment in an if-else expression.

What is the final value of numItems?

1)

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

Check Show answer

2)

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

Check Show answer

CHALLENGE ACTIVITY 3.4.3: If-else statement: Fix errors.

Start

Find and fix the error in the if-else statement.

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     int userNum;
6
7     cin >> userNum;    // Program will be tested with values: 1, 2, 3, 4.
8
9     if (userNum = 2) {
10       cout << "Num is less or equal to two" << endl;
11    }
12    else {
13       cout << "Num is greater than two" << endl;
14    }
15
16    return 0;
17 }
```

1

Check Try again

CHALLENGE ACTIVITY 3.4.4: If-else statement: Print senior citizen.

Write an if-else statement that checks patronAge. If 55 or greater, print "Senior citizen", otherwise print "Not senior citizen" (without quotes). End with newline.

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

int main() {
   int patronAge;

   cin >> patronAge;

   /* Your solution goes here  */

   return 0;
}
```


Run

View your last submission

3.5 Detecting ranges (general)

Range detection using if-elseif-else

An if-elseif-else structure can elegantly detect number ranges, such as under 6, 6 - 7, 8 - 9, 10 - 11, and 12 and up, with each branch performing a different action for each range. Each expression only needs to indicate the upper range part; if execution reaches an expression, the lower range part is implicit from the previous expressions being false.

PARTICIPATION ACTIVITY 3.5.1: An if-elseif-else structure can elegantly detect ranges.

Animation captions:

1. Kids of various ages may wish to play soccer. A soccer club may not have teams for kids 5 and under.

2. One level of teams is listed as "Under 8" (or just U8), which is understood to mean just 7 or 6, but not 5 or younger.
3. Likewise, U10 means 9 and 8, and U12 means 11 and 10. No teams exist for ages 12 and over.
4. An if-elseif-else structure can elegantly capture such ranges. When an expression is checked, one knows that all the previous expressions were false, thus defining the low range end.

PARTICIPATION ACTIVITY 3.5.2: Using if-elseif-else to detect increasing ranges.

Indicate the range corresponding to each branch. x is a non-negative integer.

20 - 29 **0 - 9** **30+** **10 - 19**

	If x < 10 : Branch 1
	Else If x < 20 : Branch 2
	Else If x < 30 : Branch 3
	Else : Branch 4

Reset

PARTICIPATION ACTIVITY 3.5.3: More ranges with if-elseif-else.

Indicate the range detected by the expression, assuming each question continues a single if-elseif-else structure. x is an integer. Type ranges as: 25 - 29

1) If x > 100 : Branch 1

[ ] - infinity

Check **Show answer**

2) Else If x > 50 : Branch 2

[ ]

Check **Show answer**

3) Else

-infinity -

Check Show answer

4) Is this a reasonable if-elseif-else structure? Type yes or no.

If x < 100: Branch 1
Else If x < 200: Branch 2
Else If x < 150: Branch 3
Else: Branch 4

Check Show answer

3.6 Detecting ranges with if-else statements

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 < 16 *is false* (so 16 or greater) AND userAge is < 25, meaning userAge is between 16 - 24 (inclusive).

Figure 3.6.1: Using sequential nature of multi-branch if-else for ranges: Insurance prices.

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

int main() {
   int userAge;
   int insurancePrice;

   cout << "Enter your age: ";
   cin  >> userAge;

   if (userAge < 16) {                  // Age 15 and under
      cout << "Too young." << endl;
      insurancePrice = 0;
   }
   else if (userAge < 25) {             // Age 16 - 24
      insurancePrice = 4800;
   }
   else if (userAge < 40) {             // Age 25 - 39
      insurancePrice = 2350;
   }
   else {                                // Age 40 and up
      insurancePrice = 2100;
   }

   cout << "Annual price: $" << insurancePrice << endl;

   return 0;
}
```

```
Enter your age: 19
Annual price: $4800

...

Enter your age: 27
Annual price: $2350
...

Enter your age: 15
Too young.
Annual price: $0

...

Enter your age: 129
Annual price: $2100
```


Source: carsdirect.com, 2017

PARTICIPATION ACTIVITY 3.6.1: Ranges and multi-branch if-else.

Type the range for each branch. Type ranges as: 25 - 29, or type 30+ for all numbers 30 and larger.

```
if (numSales < 10) {
   ...
}
else if (numSales < 20) {  // 2nd branch range: _____
   ...
}
else if (numSales < 30) {  // 3rd branch range: _____
   ...
}
else {                     // 4th branch range: _____
   ...
}
```


1) 2nd branch range:

Check Show answer

2) 3rd branch range:

Check Show answer

3) 4th branch range:

Check Show answer

4) What is the range for the last branch below?

```
if (numItems < 0) {
   ...
}
else if (numItems > 100) {
   ...
}
else {  // Range: _____
   ...
}
```

Check Show answer

PARTICIPATION ACTIVITY 3.6.2: Complete the multi-branch if-else.

1) Second branch: userNum is less than 200

```
if (userNum < 100 ) {

   ...

}

else if (                    )
{

   ...

}

else { // userNum >= 200

   ...

}
```


Check Show answer

2) Second branch: userNum is positive (non-zero)

```
if (userNum < 0 ) {

   ...

}

                    {

   ...

}

else { // userNum is 0

   ...

}
```


Check Show answer

3) Second branch: userNum is greater than 105

```
if (userNum < 100 ) {

   ...

}
```

[ ] {


```
   ...

}

else { // userNum is between

      // 100 and 105

   ...

}
```

Check Show answer

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?
}
```

Check Show answer

5) 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
}
```

Check Show answer

CHALLENGE ACTIVITY 3.6.1: Detect ranges using branches.

CHALLENGE ACTIVITY 3.6.2: Multi-branch if-else statement: Print century.

Write an if-else statement with multiple branches. If givenYear is 2101 or greater, print "Distant future" (without quotes). Else, if givenYear is 2001 or greater (2001-2100), print "21st century". Else, if givenYear is 1901 or greater (1901-2000), print "20th century". Else (1900 or earlier), print "Long ago". End with a newline. Remember that outputting 'endl' outputs a newline.

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

int main() {
   int givenYear;

   cin >> givenYear;

   /* Your solution goes here  */

   return 0;
}
```

Run

View your last submission

3.7 Logical operators

Logical AND, OR, and NOT (general)

A ***logical operator*** treats operands as being true or false, and evaluates to true or false. Logical operators include AND, OR, and NOT. Programming languages typically use various symbols for those operators, but below the words AND, OR, and NOT are used for introductory purposes.

PARTICIPATION ACTIVITY 3.7.1: Logical operators: AND, OR, and NOT.

Animation captions:

1. AND evaluates to true only if BOTH operands are true.
2. OR evaluates to true if ANY operand is true (one, the other, or both).
3. NOT evaluates to the opposite of the operand.
4. Each operand is commonly an expression itself. If x = 7, y = 9, then (x > 0) AND (y < 10) is true AND true, so evaluates to true (both operands are true).

Table 3.7.1: Logical operators.

Logical operator	Description
a AND b	***Logical AND***: true when both of its operands are true
a OR b	***Logical OR***: true when at least one of its two operands are true
NOT a	***Logical NOT***: true when its one operand is false, and vice-versa.

PARTICIPATION ACTIVITY 3.7.2: Evaluating expressions with logical operators.

Indicate whether the expression evaluates to true or false.
x is 7, y is 9.

1) x > 5

- ○ true
- ○ false

2) (x > 5) AND (y < 20)

- ○ true
- ○ false

3) (x > 10) AND (y < 20)

- true
- false

4) (x > 10) OR (y < 20)

- true
- false

5) (x > 10) OR (y > 20)

- true
- false

6) NOT (x > 10)

- true
- false

7) NOT ((x > 5) AND (y < 20))

- true
- false

Detecting ranges with logical operators (general)

A common use of logical operators is to detect if a value is within a range.

PARTICIPATION ACTIVITY 3.7.3: Using AND to detect if a value is within a range.

Animation captions:

1. The range 10 < x < 15 means that x may be 11, 12, 13, 14.
2. Specifying that range in a program can be done using two < operators along with an AND operator. 10 < x defines the range 11 and higher.
3. x < 15 defines the range 14 and lower. ANDing yields the overlapping range. Only when x is 11, 12, 13, or 14 will both expressions be true.

PARTICIPATION ACTIVITY 3.7.4: Using AND to detect if a value is within a range.

1) Which approach uses a logical operator to detect if x is in the range 1 to 99.
   - ○ 0 < x < 100
   - ○ (0 < x) AND (x < 100)
   - ○ (0 < x) AND (x > 100)

2) Which detects if x is in the range -4 to +4?
   - ○ (x < -5) AND (x < 5)
   - ○ (x > -5) OR (x < 5)
   - ○ (x > -5) AND (x < 5)

3) Which detects if x is either less than -5, or greater than 10?
   - ○ (x < -5) AND (x > 10)
   - ○ (x < -5) OR (x > 10)

Logical operators

Special symbols are used to represent the AND, OR, and NOT logical operators.

Table 3.7.2: 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*** (!): true when its one operand is false, and vice-versa.

PARTICIPATION ACTIVITY 3.7.5: Logical operators.

Match the symbol with the logical operator.

!! ! && ||

	AND
	OR
	NOT
	No such operator

Reset

PARTICIPATION ACTIVITY 3.7.6: Evaluating expressions with logical operators.

Given numPeople = 10, userKey = 'q'. Indicate whether the expression evaluates to true or false.

1) `(numPeople >= 10) && (userKey == 'x')`
   - true
   - false

2) `(numPeople >= 20) || (userKey == 'q')`
   - true
   - false

3) `!(userKey == 'a')`
   - true
   - false

4) `!( (numPeople == 5) || (numPeople == 6) )`
   - true
   - false

Logical operators are commonly used in expressions found in if-else statements.

PARTICIPATION ACTIVITY 3.7.7: Logical operators: Complete the expressions to detect the desired range.

1)

daysLogged is greater than 30 and less than 90

```
if ( (daysLogged > 30)
(daysLogged < 90) ) {
   ...
}
```

Check Show answer

2) 0 < maxCars < 100

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

Check Show answer

3) numStores is between 10 and 20, inclusive.

```
if ( (numStores >= 10)
(numStores <= 20) ) {
   ...
}
```

Check Show answer

4) notValid is either less than 15, or greater than 79.

```
if ( (notValid < 15)
(notValid > 79) ) {
   ...
}
```

Check Show answer

PARTICIPATION ACTIVITY 3.7.8: Creating expressions with logical operators.

1) numDogs has a minimum of 2 and a maximum of 5.

```
if ( (numDogs >= 2)
                        ) {
    ...
}
```

Check Show answer

2) wage is greater than 10 and less than 18. Use > and < (not >= and <=). Use parentheses around sub-expressions.

```
if (                        ) {
    ...
}
```

Check Show answer

3) num is a 3-digit positive integer. Ex: 100, 989, and 523, are 3-digit positive integers, but 55, 1000, and -4 are not.

For most direct readability, your expression should compare directly with the smallest and largest 3-digit number.

```
if ( (num >= 100)
                        ) {
    ...
}
```

Check Show answer

PARTICIPATION ACTIVITY 3.7.9: Logical expression simulator.

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

Example: TV channels

A cable TV provider may have regular channels numbered 2-499, and high-definition channels numbered 1002-1499. A program may set a character variable to 's' for standard, 'h' for high-definition, and 'e' for error.

Figure 3.7.1: Detecting ranges: Cable TV channels.


```
if ( (userChannel >= 2) && (userChannel <= 499) ) {
   channelType = 's';
}
else if ((userChannel >= 1002) && (userChannel <= 1499) ) {
   channelType = 'h';
}
else {
   channelType = 'e';
}
```

zyDE 3.7.1: Detecting ranges: Cable TV channels.

Run the program and observe the output. Change the input box value from 3 to another number, and run again.

Load default template...

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     int userChannel;
6     char channelType;
7
8     cin  >> userChannel;
9
10    if ( (userChannel >= 2) && (userChar
11       channelType = 's';
12    }
13    else if ((userChannel >= 1002) && (u
14       channelType = 'h';
15    }
16    else {
17
```

3

Run

PARTICIPATION ACTIVITY 3.7.10: TV channel example: Detecting ranges.

Consider the above example.

1) If userChannel is 300, to what does the if statement's expression, `(userChannel >= 2) && (userChannel <= 499)`, evaluate?
   - ○ true
   - ○ false

2) If userChannel is 300, does the else if's expression `(userChannel >= 1002) && (userChannel <= 1499)` get checked?
   - ○ Yes
   - ○ No

3) Did the expressions use logical AND or logical OR?
   - ○ AND
   - ○ OR

4) Channels 500-599 are pay channels. Does this expression detect that range? `(userChannel >= 500) || (userChannel <= 599)`
   - ○ Yes
   - ○ No

Detecting ranges implicitly vs. explicitly

If a program should detect increasing ranges without gaps, a multi-branch if-else statement can be used without logical operators; the low-end of the range is implicitly known upon reaching an expression. Likewise, a decreasing range without gaps has implicitly-known high-ends. In contrast, when gaps exist, the range's low and high ends must both be explicitly detected, using a logical operator.

PARTICIPATION ACTIVITY 3.7.11: Detecting ranges implicitly vs. explicitly.

Animation content:

`undefined`

Animation captions:

1. This code detects ranges explicitly using the AND operator. The first branch executes when x < 0, the second when (x >=0) && (x <= 10).
2. But, if the first branch doesn't execute, x must be >= 0, so the second branch's expression can just be x <= 10. The x >= 0 is implicit.
3. If the range has gaps, the range's ends must be explicitly indicated, using AND. So if the if expression is x < 0, and then next range is 16..25, an explicit range is needed.

PARTICIPATION ACTIVITY 3.7.12: Detecting ranges implicitly vs. explicitly.

Using a multi-branch if-else statement, indicate whether one end of each range can be detected implicitly, or whether both ends must be detected explicitly.

1) <= 10
11..20
21..50
51+

- Implicit
- Explicit

2) <=10
50..100
150..200

- Implicit
- Explicit

3) >= 100
90..99
80..89

- Implicit
- Explicit

CHALLENGE ACTIVITY 3.7.1: Enter the output of the Boolean expressions.

CHALLENGE ACTIVITY 3.7.2: Detect specific values.

Write an expression that prints "Special number" if specialNum is 0, -99, or 44.

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

int main() {
   int specialNum;

   cin >> specialNum;

   if (/* Your solution goes here  */) {
      cout << "Special number" << endl;
   }
   else {
      cout << "Not special number" << endl;
   }

   return 0;
}
```


Run

View your last submission

CHALLENGE ACTIVITY 3.7.3: Detect number range.

Write an expression that prints "Eligible" if userAge is between 18 and 25 inclusive.
Ex: 17 prints "Ineligible", 18 prints "Eligible".

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

int main() {
   int userAge;

   cin >> userAge;

   if (/* Your solution goes here  */) {
      cout << "Eligible" << endl;
   }
   else {
      cout << "Ineligible" << endl;
   }

   return 0;
}
```


Run

View your last submission

3.8 Example: Toll calculation

Calculating toll based on time of day

The section presents an example program that calculates the toll amount for travel along a toll road or toll lane. The toll amount is based on the time of day, day of the week, and number of persons in the vehicle.

The initial version of the program calculates the toll amount for travel on a weekday based upon the toll schedule below. The table lists times in both am/pm format and 24-hour format.

Table 3.8.1: Weekday toll schedule.

Time (am/pm)	Time (24 hour)	Toll amount
Before 6:00 am	Before 6:00	1.55
6:00 am to 9:59 am	6:00 to 9:59	4.65
10:00 am to 5:59 pm	10:00 to 17:59	2.35
6:00 pm and after	18:00 and after	1.55

The program gets the time of travel from the user using 24 hours format, and uses the hour to determine the toll amount. A multi-branch if-else statement is used to determine in which range the hour belongs and assigns tollAmount with the toll based on the table above, and outputs the toll.

Figure 3.8.1: Calculating toll based on time of day.

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
   int timeHour;       // Time of travel hour
   int timeMinute;     // Time of travel minute
   char inputColon;    // Used to read time format
   double tollAmount;

   cout << "Enter time of travel (HH:MM in 24 hour format): ";

   // Read an integer (hour), colon (char), and integer (minute)
   cin  >> timeHour >> inputColon >> timeMinute;

   // Determine toll based on hour of travel
   if (timeHour < 6) {          // Before 6:00 am
      tollAmount = 1.55;
   }
   else if (timeHour < 10) {    // 6 am to 9:59 am
      tollAmount = 4.65;
   }
   else if (timeHour < 18) {    // 10 am to 5:59 pm
      tollAmount = 2.35;
   }
   else {                       // 6 pm and after
      tollAmount = 1.55;
   }

   // Output time and toll amount
   cout << "Toll at " << timeHour << ":";

   // Output minute with formatting (discussed elsewhere) to
   // print two digits for minutes.
   cout << setw(2) << setfill('0') << timeMinute;
   cout << " is " << tollAmount << endl;

   return 0;
}
```



```
Enter time of travel (HH:MM in 24 hour format): 9:30
Toll at 9:30 is 4.65
```

PARTICIPATION ACTIVITY 3.8.1: Toll calculation.

For the given input, what is the final value of tollAmount?

1) 5:45

- ○ 0.00
- ○ 1.55
- ○ 2.35

2) 9:45

- 1.55
- 2.35
- 4.65

3) 10:00

- 1.55
- 2.35
- 4.65

4) 22:15

- 1.55
- 2.35

Calculating toll based on time of day and day of week

A toll road often has a different toll schedule for weekends and holidays than for weekdays. The table below lists the toll schedule for weekends and holidays.

Table 3.8.2: Toll schedule for weekends and holidays.

Time (am/pm)	Time (24 hour)	Toll amount
Before 8:00 am	Before 8:00	1.55
8:00 am to 11:59 am	8:00 to 11:59	3.05
12:00 pm to 3:59 pm	12:00 to 15:59	3.45
4:00 pm to 6:59 pm	16:00 to 18:59	3.60
7:00 pm to 9:59 pm	19:00 to 21:59	3.05
10:00 pm and after	22:00 and after	1.55

The revised program below additionally gets the type of day from the user (0 for weekdays, and 1 for weekends or holidays). The program uses nested if-else statements to calculate the toll amount. The outer if-else checks if the today is a weekday or weekend/holiday. The nested if-else statements implement the respective toll schedules by determining the appropriate toll based on the hour of travel.

The program also uses if-else statements to output the time of travel using am/pm format instead of 24-hour format.

Figure 3.8.2: Calculating toll based on time of day and day of week.

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
   int timeHour;       // Time of travel hour (24 hour format)
   int timeMinute;     // Time of travel minute
   int typeOfDay;      // 0 - weekday, 1 - weekend/holiday
   char inputColon;    // Used to read time format
   double tollAmount;

   cout << "Enter time of travel (HH:MM in 24 hour format): ";

   // Read an integer (hour), colon (char), and integer (minute)
   cin  >> timeHour >> inputColon >> timeMinute;

   cout << "Enter type of day (0 - weekday, 1 - weekend/holiday): ";
   cin >> typeOfDay;

   if (typeOfDay == 0) { // Weekday time and rates
                         // Determine toll based on hour of travel
      if (timeHour < 6) {          // Before 6:00 am
         tollAmount = 1.55;
      }
      else if (timeHour < 10) {    // 6 am to 9:59 am
         tollAmount = 4.65;
      }
      else if (timeHour < 18) {    // 10 am to 5:59 pm
         tollAmount = 2.35;
      }
      else {                       // 6 pm and after
         tollAmount = 1.55;
      }
   }
   else { // Weekend/holiday time and rates
          // Determine toll based on hour of travel
      if (timeHour < 8) {          // Before 8:00 am
         tollAmount = 1.55;
      }
      else if (timeHour < 12) {    // 8 am to 11:59 am
         tollAmount = 3.05;
      }
      else if (timeHour < 16) {    // 12 pm to 3:59 pm
         tollAmount = 3.45;
      }
      else if (timeHour < 19) {    // 4 pm to 6:5 9pm
         tollAmount = 3.60;
      }
      else if (timeHour < 22) {    // 7 pm to 9:59 pm
         tollAmount = 3.05;
      }
      else {                       // 10 pm and after
         tollAmount = 1.55;
      }
   }

   // Output toll using am/pm format
   cout << "Toll at ";

   // Output hour adjusting for am/pm format
```

```
   if (timeHour == 0) {
      cout << "12:";
   }
   else if (timeHour <= 12) {
      cout << timeHour << ":";
   }
   else {
      cout << timeHour - 12 << ":";
   }

   // Output minute with formatting (discussed elsewhere) to
   // print two digits for minutes.
   cout << setw(2) << setfill('0') << timeMinute;

   // Output am/pm
   if( timeHour < 12 ) {
      cout << " am";
   }
   else {
      cout << " pm";
   }

   cout << " is " << tollAmount << endl;

   return 0;
}
```



```
Enter time of travel (HH:MM in 24 hour format): 10:45
Enter type of day (0 - weekday, 1 - weekend/holiday): 1
Toll at 10:45 am is 3.05
```

PARTICIPATION ACTIVITY 3.8.2: If-else statements for calculating toll amount and formatting time.

1) The outer if-else statement checks the type of day, and the nested if-else statements check the hour of travel.
   - True
   - False

2) An alternative implementation that checks the hour of travel in an outer if-else statements and checks the type of day using nested if-else statements would have the same number of if statements.
   - True
   - False

3) If timeHour is 0 and timeMinute is 30, the time will be output as: 0:30.
   - True

False

Calculating toll with carpool discount

A toll road may have a discount for carpools, sometimes called high-occupancy vehicles (HOV). The following program uses if-else statement to adjust the toll amount based on the number of persons in the vehicle. The carpool discount rules are:

- A carpool is 3 or more person per vehicle.
- The toll for carpools on weekdays between 6:00 am and 10:00 am is half the normal toll.
- Otherwise, the toll for carpools is 0 (as in free).

Figure 3.8.3: Calculating toll with carpool discount.

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
   int timeHour;       // Time of travel hour (24 hour format)
   int timeMinute;     // Time of travel minute
   int typeOfDay;      // 0 - weekday, 1 - weekend/holiday
   int numPersons;     // Persons in vehicle
   char inputColon;    // Used to read time format
   double tollAmount;

   cout << "Enter time of travel (HH:MM in 24 hour format): ";

   // Read an integer (hour), colon (char), and integer (minute)
   cin  >> timeHour >> inputColon >> timeMinute;

   cout << "Enter type of day (0 - weekday, 1 - weekend/holiday): ";
   cin >> typeOfDay;

   cout << "Enter number of persons in vehicle: ";
   cin >> numPersons;

   if (typeOfDay == 0) { // Weekday time and rates
                         // Determine toll based on hour of travel
      if (timeHour < 6) {          // Before 6:00 am
         tollAmount = 1.55;
      }
      else if (timeHour < 10) {    // 6 am to 9:59 am
         tollAmount = 4.65;
      }
      else if (timeHour < 18) {    // 10 am to 5:59 pm
         tollAmount = 2.35;
      }
      else {                       // 6 pm and after
         tollAmount = 1.55;
      }
   }
   else { // Weekend/holiday time and rates
          // Determine toll based on hour of travel
      if (timeHour < 8) {          // Before 8:00 am
         tollAmount = 1.55;
      }
      else if (timeHour < 12) {    // 8 am to 11:59 am
         tollAmount = 3.05;
      }
```

```
      }
      else if (timeHour < 16) {    // 12 pm to 3:59 pm
         tollAmount = 3.45;
      }
      else if (timeHour < 19) {    // 4 pm to 6:59 pm
         tollAmount = 3.60;
      }
      else if (timeHour < 22) {    // 7 pm to 9:59 pm
         tollAmount = 3.05;
      }
      else {                       // 10 pm and after
         tollAmount = 1.55;
      }
   }

   // Check for carpool rate (3 or more persons) and update toll
   if (numPersons >= 3) {
      // If on a weekday between 6:00 am and 9:59 am, toll is half off
      if ((typeOfDay == 0) && (timeHour >= 6) && (timeHour < 10)) {
         tollAmount = tollAmount * 0.5;
      }
      // Otherwise, the toll is free
      else {
         tollAmount = 0.0;
      }
   }

   // Output toll using am/pm format
   cout << "Toll at ";

   // Output hour adjusting for am/pm format
   if (timeHour == 0) {
      cout << "12:";
   }
   else if (timeHour <= 12) {
      cout << timeHour << ":";
   }
   else {
      cout << timeHour - 12 << ":";
   }

   // Output minute with formatting (discussed elsewhere) to
   // print two digits for minutes.
   cout << setw(2) << setfill('0') << timeMinute;

   // Output am/pm
   if( timeHour < 12 ) {
      cout << " am";
   }
   else {
      cout << " pm";
   }

   cout << " is " << tollAmount << endl;

   return 0;
}
```



```
Enter time of travel (HH:MM in 24 hour format): 17:15
Enter type of day (0 - weekday, 1 - weekend/holiday): 0
Enter number of persons in vehicle: 3
Toll at 5:15 pm is 0
```


PARTICIPATION ACTIVITY 3.8.3: Toll calculation.

Match the final value of tollAmount to the timeHour, typeOfDay, and numPeople.

1.55 | 0.0 | 2.325 | 4.65

	timeHour is 7, typeOfDay is 0, numPeople is 1
	timeHour is 8, typeOfDay is 0, numPeople is 4
	timeHour is 18, typeOfDay is 1, numPeople is 3
	timeHour is 20, typeOfDay is 0, numPeople is 2

Reset

3.9 Order of evaluation

Precedence rules

The order in which operators are evaluated in an expression are known as ***precedence rules***. Arithmetic, logical, and relational operators are evaluated in the order shown below.

Table 3.9.1: Precedence rules for arithmetic, logical, and relational operators.

Operator/Convention	Description	Explanation
( )	Items within parentheses are evaluated first	In `(a * (b + c)) - d`, the + is evaluated first, then *, then -.
!	! (logical NOT) is next	`! x || y` is evaluated as `(!x) || y`

* / % + -	Arithmetic operators (using their precedence rules; see earlier section)	`z - 45 * y < 53` evaluates * first, then -, then <.
`< <= > >=`	Relational operators	`x < 2 || x >= 10` is evaluated as `(x < 2) || (x >= 10)` because < and >= have precedence over ||.
`== !=`	Equality and inequality operators	`x == 0 && x >= 10` is evaluated as `(x == 0) && (x >= 10)` because < and >= have precedence over &&. == and != have the same precedence and are evaluated left to right.
&&	Logical AND	`x == 5 || y == 10 && z != 10` is evaluated as `(x == 5) || ((y == 10) && (z != 10))` because && has precedence over ||.
||	Logical OR	|| has the lowest precedence of the listed arithmetic, logical, and relational operators.

PARTICIPATION ACTIVITY 3.9.1: Applying the precedence rules to an expression can be thought of as a 'tree'.

Animation content:

`undefined`

Animation captions:

1. Expressions like x + 1 > y * z || z == 3 are evaluated using precedence rules. Among +, >, *, ||, and ==, the * comes first.
2. Next comes +, then >, then ==, and finally ||.
3. The expression is actually treated like a "tree", evaluated from the bottom upwards.
4. If x is 7, y is 6, and z is 3, then y * z is 18. Next, x + 1 is 8. Next, 8 > 18 is false. Next, z == 3 is true. Finally, false || true is true.

PARTICIPATION ACTIVITY 3.9.2: Order of evaluation.

To teach precedence rules, these questions intentionally omit parentheses; good style would use parentheses to make order of evaluation explicit.

1) Which operator is evaluated first?
   ! y && x
   - ○ &&
   - ○ !

2) Which operator has precedence?
   w + 3 > x - y * z
   - ○ +
   - ○ -
   - ○ >
   - ○ *

3) In what order are the operators evaluated?
   w + 3 != y - 1 && x
   - ○ +, !=, -, &&
   - ○ +, -, &&, !=
   - ○ +, -, !=, &&

4) To what does this expression evaluate, given int x = 4, int y = 7.
   x == 3 || x + 1 > y
   - ○ true
   - ○ false

Common error: Missing parentheses

A common error is to write an expression that is evaluated in a different order than expected. Good practice is to use parentheses in expressions to make the intended order of evaluation explicit. Several examples are below.

PARTICIPATION ACTIVITY 3.9.3: Common errors in expressions.

1) Does `! x == 3` evaluate as `!(x == 3)`?

- Yes
- No

2) Does `w + x == y + z` evaluate as `(w + x) == (y + z)`?

- Yes
- No

3) Does `w && x == y && z` evaluate as `(w && x) == (y && z)`?

- Yes
- No

PARTICIPATION ACTIVITY 3.9.4: Order of evaluation.

Which illustrates the actual order of evaluation via parentheses?

1) `! green == red`

- (!green) == red
- !(green == red)
- (!green =)= red

2) `bats < birds || birds < insects`

- ((bats < birds) || birds) < insects
- bats < (birds || birds) < insects
- (bats < birds) || (birds < insects)

3) `! (bats < birds) || (birds < insects)`

- ! ((bats < birds) || (birds < insects))
- (! (bats < birds)) || (birds < insects)
- ((!bats) < birds) || (birds < insects)

4) `(num1 == 9) || (num2 == 0) && (num3 == 0)`

- (num1 == 9) || ((num2 == 0) && (num3 == 0))

- ((num1 == 9) || (num2 == 0)) && (num3 == 0)
- (num1 == 9) || (num2 == (0 && num3) == 0)

Common error: Math expression for range

A common error often made by new programmers is to write expressions like `(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, true is actually 1, and evaluating `1 < 25` will yield true. Thus, the above expression evaluates to true, even for ages greater than 25.

Thus, `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. The correct way to do such a comparison is: `(age > 16) && (age < 25)`.

PARTICIPATION ACTIVITY 3.9.5: Expression for detecting a range.

1) A programmer erroneously wrote an expression as: 0 < x < 10. Rewrite the expression using logical AND. Use parentheses.

`(0 < x)` [ ]

Check Show answer

Common error: Bitwise rather than logical operators

Logical AND is && and not just &, and logical OR is || and not just |. & and | represent ***bitwise operators***, which perform AND or OR on corresponding individual bits of the operands.

A common error is to use a bitwise operator instead of a logical operator, typing & instead of &&, or typing | instead of ||. A bitwise operator may yield different behavior than expected.

PARTICIPATION ACTIVITY 3.9.6: Bitwise vs. logical operators.

Indicate if the expression correctly uses logical operators.

1) (x > 5) & (y > 3) & (z != 0)

- Yes
- No

2) (x == 0) || (y == 0) | (z == 0)

- Yes
- No

3) ((x == y) && (y == z)) || (w == 0)

- Yes
- No

3.10 Switch statements

Switch statement

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.

PARTICIPATION ACTIVITY 3.10.1: Switch statement.

Animation captions:

1. A switch statement can more clearly represent multi-branch behavior involving a variable being compared to constant values.
2. 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.
3. If no case matches, then the default case statements are executed.

PARTICIPATION ACTIVITY 3.10.2: 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;
}
```


1) userVal = 3;

Check Show answer

2) userVal = 0;

Check Show answer

3) userVal = 2;

Check Show answer

Multi-branch if-else 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.

```
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
}
```

Switch statement general form

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.

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).

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.

Construct 3.10.1: Switch statement general form.

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

   case constantExpr2:
      // Statements
      break;

   ...

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

Figure 3.10.1: Switch example: Estimates a dog's age in human years.

```
Enter dog's age (in years): 4
That's 32 human years.

...

Enter dog's age (in years): 17
Human years unknown.
```

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

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

int main() {
   int dogAgeYears;

   cout << "Enter dog's age (in years): ";
   cin  >> dogAgeYears;

   switch (dogAgeYears) {
      case 0:
         cout << "That's 0..14 human years." << endl;
         break;

      case 1:
         cout << "That's 15 human years." << endl;
         break;

      case 2:
         cout << "That's 24 human years." << endl;
         break;

      case 3:
         cout << "That's 28 human years." << endl;
         break;

      case 4:
         cout << "That's 32 human years." << endl;
         break;

      case 5:
         cout << "That's 37 human years." << endl;
         break;

      default:
         cout << "Human years unknown." << endl;
         break;
   }

   return 0;
}
```


zyDE 3.10.1: Switch statement: Numbers to words.

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

Load default template...

```
1 #include <iostream>
2 using namespace std;
```

7

Run

```
/* Estimates dog's age in equivalent hu
   Source: www.dogyears.com
*/

int main() {
   int dogAgeYears;

   cout << "Enter dog's age (in years):
   cin  >> dogAgeYears;

   switch (dogAgeYears) {
      case 0:
         cout << "That's 0..14 human ye
         break;
```


Omitting the break statement

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 statement executes.

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

Figure 3.10.2: Switch example: Dog years with months.

```
Enter dog's age (in years): 0
Enter dog's age in months: 7
That's 5..9 human years.

...

Enter dog's age (in years): 4
FIXME: Do earlier dog year cases.
```

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

int main() {
   int dogAgeYears;
   int dogAgeMonths;

   cout << "Enter dog's age (in years): ";
   cin >> dogAgeYears;

   if (dogAgeYears == 0) {
      cout << "Enter dog's age in months: ";
      cin  >> dogAgeMonths;

      switch (dogAgeMonths) {
         case 0:
         case 1:
         case 2:
            cout << "That's 0..14 human months." << endl;
            break;

         case 3:
         case 4:
         case 5:
         case 6:
            cout << "That's 1..5 human years." << endl;
            break;

         case 7:
         case 8:
            cout << "That's 5..9 human years." << endl;
            break;

         case 9:
         case 10:
         case 11:
         case 12:
            cout << "That's 9..15 human years." << endl;
            break;

         default:
            cout << "Invalid input." << endl;
            break;
      }
   }
   else {
      cout << "FIXME: Do earlier dog year cases." << endl;
      switch (dogAgeYears) {
      }
   }

   return 0;
}
```


©zyBooks 04/25/21 07:20 488201
xiang zhao
BAYLORCSI14301440Spring2021

PARTICIPATION ACTIVITY 3.10.3: 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;
}
```


1) userChar = 'A'

Check Show answer

2) userChar = 'B'

Check Show answer

3) userChar = 'C'

Check Show answer

4) userChar = 'E'

Check Show answer

5) userChar = 'G'

Check Show answer

CHALLENGE ACTIVITY 3.10.1: Rock-paper-scissors.

Write a switch statement that checks nextChoice. If 0, print "Rock". If 1, print "Paper". If 2, print "Scissors". For any other value, print "Unknown". End with newline.


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

int main() {
   int nextChoice;

   cin >> nextChoice;

   /* Your solution goes here  */

   return 0;
}
```

Run

View your last submission

CHALLENGE ACTIVITY 3.10.2: Switch statement to convert letters to Greek letters.

Write a switch statement that checks origLetter. If 'a' or 'A', print "Alpha". If 'b' or 'B', print "Beta". For any other character, print "Unknown". Use fall-through as appropriate. End with newline.


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

int main() {
   char origLetter;

   cin >> origLetter;

```

```
9     /* Your solution goes here  */
10
11    return 0;
12 }
```


Run

View your last submission

3.11 Boolean data type

Boolean data type

Boolean refers to a quantity that has only two possible values, true or false. C++ has the built-in data type ***bool*** for representing Boolean quantities.

A Boolean variable may be set using true or false keywords. Ex: `isMale = true;` assigns isMale with the Boolean value true. A Boolean variable may also be set to the result of a logical expression. Ex: `isOverweight = (userBmi >= 25);` assigns isOverweight with the result of the expression userBmi >= 25.

Figure 3.11.1: Variables of bool data type: Life expectancy calculator.

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

int main() {
   double lifeExpectancy;
   double userBmi;
   char userChoice;
   bool isOverweight;
   bool isMale;

   // Get user's sex
   cout << "Female (f) or male (m): ";
   cin >> userChoice;
   if (userChoice == 'm') {
      isMale = true;
   }
   else {
      isMale = false;
   }

   // Get user's BMI
   cout << "Enter body mass index (BMI): ";
   cin  >> userBmi;
   isOverweight = (userBmi >= 25);

   // Determine life expectancy based on sex and BMI
   if (isMale && !isOverweight ) {
      lifeExpectancy = 79.4;
   }
   else if (!isMale && !isOverweight ) {
      lifeExpectancy = 83.5;
   }
   else if (isMale && isOverweight) {
      lifeExpectancy = 77.3;
   }
   else {
      lifeExpectancy = 81.4;
   }

   cout << "Life expectancy is " << lifeExpectancy
        << " years." << endl;

   return 0;
}
```



```
Female (f) or male (m): f
Enter body mass index (BMI): 24.1
Life expectancy is 83.5 years.

...

Female (f) or male (m): m
Enter body mass index (BMI): 28.2
Life expectancy is 77.3 years.
```

Source: Life expectancy calculator (Bankrate.com, 2020)

PARTICIPATION ACTIVITY 3.11.1: Boolean variables.

1) Write a statement that declares a Boolean variable named hasHighBP.

Check Show answer

2) Write a statement that assigns hasHighBP with false.

Check Show answer

3) isSunny, isDayOff, and isBeachDay are Boolean variables. What is isBeachDay after executing the following statements? Type true or false.

```
isSunny = false;
isDayOff = true;
isBeachDay = false;

if (isSunny && isDayOff) {
   isBeachDay = true;
}
```

Check Show answer

Uses of Boolean data types

A programmer can use a Boolean variable to simplify a complex expression. An expression that combines logical and relational operators can be simplified by assigning bool variables with the result of the expression using relational operators. The if-else expression can then consist of only logical operations using those variables.

The following program assigns bool variables isHot, isReallyHot, and isHumid with the results of expressions comparing currentTemp, desiredTemp, and currentHumidity. The if-else statement then uses isHot and isHumid in the if-else's expressions.

Figure 3.11.2: Using Boolean variables to simplify expressions.

```
isHot = (currentTemp > desiredTemp);
isReallyHot = (currentTemp > (desiredTemp + 5.0));
isHumid = (currentHumidity > 0.50);

if (isReallyHot) {
   // Use A/C and evaporative cooler
   acOn = true;
   evapCoolerOn = true;
}
else if (isHot && isHumid) {
   // Use A/C
   acOn = true;
   evapCoolerOn = false;
}
else if (isHot && !isHumid) {
   // Use evaporative cooler
   acOn = false;
   evapCoolerOn = true;
}
else {
   acOn = false;
   evapCoolerOn = false;
}
```


PARTICIPATION ACTIVITY 3.11.2: Simplifying expressions.

Given the following if-else statement:

```
if ( (userAge > 13) && (userAge < 21) && studentGpa >= 3.5 ) {
   studentDiscount = 7.5;
}
else if ( (userAge > 13) && (userAge < 21) && studentGpa >= 2.75 ) {
  studentDiscount = 5.0;
}
else {
  studentDiscount = 0.0;
}
```

1) Write a statement that assigns the variable veryGoodGpa with an expression that evaluates to true if studentGpa is greater than or equal to 3.5.

```
veryGoodGpa = (
                                    );
```

Check Show answer

2) Write a statement that assigns the

variable goodGpa with an expression that evaluates to true if studentGpa is greater than 2.75.

```
goodGpa = (
                    );
```

Check Show answer

3) Write a statement that assigns the variable isInAgeRange with an expression that evaluates to true if userAge is greater than 13 and less than 21.

```
isInAgeRange = (
                    );
```

Check Show answer

4) Revise the if expression above to use the variables isInAgeRange and veryGoodGpa.

```
if (
                    )
{
   studentDiscount = 7.5;
}
```

Check Show answer

CHALLENGE ACTIVITY 3.11.1: Using bool.

Assign isTeenager with true if kidAge is 13 to 19 inclusive. Otherwise, assign isTeenager with false.


```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5    bool isTeenager;
6    int kidAge;
7
```

```
   cin >> kidAge;

   /* Your solution goes here  */

   if (isTeenager) {
      cout << "Teen" << endl;
   }
   else {
      cout << "Not teen" << endl;
   }
```


Run

View your last submission

CHALLENGE ACTIVITY 3.11.2: Bool in branching statements.

Write an if-else statement to describe an object. Print "Balloon" if isBalloon is true and isRed is false. Print "Red balloon" if isBalloon and isRed are both true. Print "Not a balloon" otherwise. End with newline. (Notes)

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

int main() {
   bool isRed;
   bool isBalloon;

   cin >> isRed;
   cin >> isBalloon;

   /* Your solution goes here  */

   return 0;
}
```


Run

View your last submission

3.12 String comparisons

String comparison: Equality

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

PARTICIPATION ACTIVITY 3.12.1: Equal strings.

Which strings are equal?

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 the equality operators == and !=.

Figure 3.12.1: String equality example: Censoring.

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

int main() {
   string userWord;

   cout << "Enter a word: ";
   cin  >> userWord;

   if (userWord == "Voldemort") {
      cout << "He who must not be named";
   }
   else {
      cout << userWord;
   }
   cout << endl;

   return 0;
}
```

```
Enter a word: Sally
Sally

...

Enter a word: Voldemort
He who must not be named

...

Enter a word: voldemort
voldemort
```


PARTICIPATION ACTIVITY 3.12.2: Comparing strings for equality.

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

1) str1 == "Apples"
   - True
   - False

2) str1 == str2
   - True
   - False

3) str2 != "oranges"
   - True
   - False

String comparison: Relational

Strings are sometimes compared relationally (less than, greater than), as when sorting words alphabetically. A comparison begins at index 0 and compares each character until the evaluation

results in false, or the end of a string is reached. 'A' is 65, 'B' is 66, etc., while 'a' is 97, 'b' is 98, etc. So "Apples" is less than "apples" because 65 is less than 97.

PARTICIPATION ACTIVITY 3.12.3: String comparison.

Animation captions:

1. A string comparison compares each character. The first five characters of studentName and teacherName are the same.
2. 'J' is greater than 'A', so studentName is greater than teacherName.

PARTICIPATION ACTIVITY 3.12.4: Case matters in string comparisons.

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

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 compares strings relationally using the relational operators <, <=, >, and >=.

A common error is to forget that case matters in a string comparison. A programmer can compare strings while ignoring case by first converting both strings to lowercase before comparing (discussed elsewhere).

PARTICIPATION ACTIVITY 3.12.5: Relational string comparison.

1) Write an expression that evaluates to

true if myName is greater than yourName.

```
if (myName          yourName) {
   ...
}
```

Check Show answer

2) Write an expression that evaluates to true if authorName1 is less than or equal to authorName2.

```
if (
                                  )
{
   ...
}
```

Check Show answer

CHALLENGE ACTIVITY 3.12.1: String comparison: Detect word.

Write an if-else statement that prints "Goodbye" if userString is "Quit", else prints "Hello". End with newline.

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main() {
6     string userString;
7
8     cin >> userString;
9
10    /* Your solution goes here  */
11
12    return 0;
13 }
```


Run

View your last submission

CHALLENGE ACTIVITY 3.12.2: Print two strings in alphabetical order.

Print the two strings, firstString and secondString, in alphabetical order. Assume the strings are lowercase. End with newline. Sample output:

```
capes rabbits
```



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

int main() {
   string firstString;
   string secondString;

   cin >> firstString;
   cin >> secondString;

   /* Your solution goes here  */

   return 0;
}
```

Run

View your last submission

3.13 String access operations

String character indices

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

PARTICIPATION ACTIVITY 3.13.1: A string's characters each has an index, starting with 0.

Animation captions:

1. A string is a sequence of characters. So "Mary" is a sequence of characters M, a, r, y.
2. Each character has an index, starting with 0 (not 1). For a four-character string like "Mary", M is at index 0, a at 1, r at 2, and y at 3. A 4-character string's last index is 3, not 4.

PARTICIPATION ACTIVITY 3.13.2: String indices.

1) For string "Light", what is L's index?
   - 0
   - 1

2) For string "Light", what is i's index?
   - 0
   - 1
   - 2

3) For string "Light", what is t's index?
   - 4
   - 5

4) For string "Oh my", which character is at index 2?
   - h
   - (space)
   - m

5) For string "You wish!", which character is at index 4?
   - (space)
   - w

6) For string "You wish!", which character is at index 9?
   - h
   - !
   - "
   - No such character

Accessing string characters

at(): The notation someString.at(x) accesses the character at index x of a string.

Figure 3.13.1: String character access: Word scramble.


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

int main() {
   string userWord;

   cout << "Enter a 5-letter word: ";
   cin  >> userWord;

   cout << "Scrambled: ";
   cout << userWord.at(3);
   cout << userWord.at(1);
   cout << userWord.at(4);
   cout << userWord.at(0);
   cout << userWord.at(2);
   cout << endl;

   return 0;
}
```

```
Enter a 5-letter word: water
Scrambled: earwt

...

Enter a 5-letter word: Quick
Scrambled: cukQi

...

Enter a 5-letter word: 98765
Scrambled: 68597
```

PARTICIPATION ACTIVITY 3.13.3: Accessing string characters.

Given string userString is "Run!". Indicate char x's value.

1) x = userString.at(0);

- R
- u

2) x = userString.at(3);

- n
- !

3) x = userString.at(4);

- (blank)
- (error)

Changing a character in a string

A character in a string can be assigned. If userString is "abcde", then userString.at(3) = 'X' yields "abcXe".

Figure 3.13.2: Example: Changing a character.


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

int main() {
   string userWord ="Caterpillar";
   int replaceIndex;

   cout << "Enter an index (0-10): ";
   cin  >> replaceIndex;

   userWord.at(replaceIndex) = '*';

   cout << "Updated string: ";
   cout << userWord << endl;

   return 0;
}
```

```
Enter an index (0-10): 0
Updated string: *aterpillar

...

Enter an index (0-10): 3
Updated string: Cat*rpillar

...

Enter an index (0-10): 10
Updated string: Caterpilla*
```

PARTICIPATION ACTIVITY 3.13.4: Assigning a string character.

Given string firstName is "Ron".

1) What is firstName after:
   firstName.at(1) = '@';
   - @on
   - R@n

2) What is firstName after:
   firstName.at(3) = '!';
   - Ron!
   - (error)

3) What is firstName after:
   firstName.at(0) = "XX";
   - XXn
   - (error)

Working with the end of a string

Determining the last character in a string is often useful. If a string's length is known, the last character is at index length - 1. Ex: "Hey" has length 3, with y at index 2. The function s1.***size()*** returns s1's length. Ex: If s1 is "Hey", s1.size() returns 3.

A common task is to append (add to the end) a string to an existing string. The function s1.***append***(s2) appends string s2 to string s1. Ex: If s1 is "Hey", s1.append("!!!") makes s1 "Hey!!!".

The example program below might be part of a "caption contest" for a website where a user enters a string below an image. The program automatically adds a period if the user did not include any punctuation at the caption's end.

Figure 3.13.3: Example: Adding a period to a caption if no punctuation.

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

int main() {
   string userCaption;
   char lastChar;
   int lastIndex;

   cout << "Enter a caption: ";
   getline(cin, userCaption);

   lastIndex = userCaption.size() - 1;
   lastChar  = userCaption.at(lastIndex);

   if ( (lastChar != '.') && (lastChar != '!') && (lastChar
!= '?') ) {
      // User's caption lacked ending punctuation, so add a
period
      userCaption.append(".");
   }

   cout << "New: ";
   cout << userCaption << endl;

   return 0;
}
```

```
Enter a caption: Hello
world
New: Hello world.

...

Enter a caption: Anyone
home?
New: Anyone home?

...

Enter a caption: TGIF!
New: TGIF!

...

Enter a caption: Another
day.
New: Another day.

...

Enter a caption: Life is
sweet
New: Life is sweet.
```

size() and length()

The size() and length() functions both return a string's length. Ex: For the string firstName = "Tosi", firstName.size() and firstName.length() both return 4.

PARTICIPATION ACTIVITY 3.13.5: Working with the end of a string.

1) What is the index of the last letter in "Hey"?
   - ○ 2
   - ○ 3

2) What is the length of string "Hey"?
   - ○ 2
   - ○ 3

3) Is a string's length the same as a string's last index?
   - ○ Yes
   - ○ No, the length is 1 greater.
   - ○ No, the length is 1 less.

PARTICIPATION ACTIVITY 3.13.6: String length.

userText is "March 17, 2034".

1) What does userText.size() return?

Check Show answer

2) What is the index of the last character in userText?

Check Show answer

3) What character does userText.at(userText.size() - 1) return?

Check Show answer

PARTICIPATION ACTIVITY 3.13.7: Working with the end of a string.

For each question userString is initially "Done". Indicate userString's value after the statement provided.

Do not type quotes in your answer, so type Done rather than "Done". If appropriate, type: Error

1) userString.append("!");

Check Show answer

2) userString.append('?');

Check Show answer

3) userString.append(" now");

Check Show answer

4) anotherString = "yet...";
userString.append(anotherString);

Check Show answer

5) userString.at(userString.size()) = 't';

Check Show answer

6) userString.at(userString.size() - 1) = 't';

Check Show answer

©zyBooks 04/25/21 07:20 488201
xiang zhao
BAYLORCSI14301440Spring2021

Common errors

A common error is to access an invalid string 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.

PARTICIPATION ACTIVITY 3.13.8: Common error: Out-of-range access yields an exception.

Animation captions:

1. String variable userName has size 3 (so last index is 2). Those 3 items happen to be in memory locations 76, 77, 78. Location 79 has some other variable's value.
2. Accesses to indices 0, 1, 2 are fine. But accessing index 3 is an error.

The .at(index) function generates an exception if the index is out of range for the string's size. An ***exception*** is a detected runtime error that commonly prints an error message and terminates the program.

C++ also supports C-style access of a string using brackets [] rather than .at(), as in: someString[0]. However, such C-style access does not provide such error checking. Good practice is to use .at() rather than brackets for accessing a string's characters, due to .at()'s error checking.

PARTICIPATION ACTIVITY 3.13.9: Out-of-range string access.

userText is "Monday".

1) userText.at(7) = '!' may write to another variable's location and cause bizarre program behavior.
   - True
   - False

2) userText[7] = '!' may write to another variable's location and cause bizarre program behavior.
   - True
   - False

3) userText.at(userText.size()) yields 'y'.
   - True
   - False

CHALLENGE

ACTIVITY 3.13.1: String library functions.

Assign the size of userInput to stringSize. Ex: if userInput is "Hello", output is:

```
Size of userInput: 5
```



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

int main() {
   string userInput;
   int stringSize;

   getline(cin, userInput);

   /* Your solution goes here  */

   cout << "Size of userInput: " << stringSize << endl;

   return 0;
}
```

Run

View your last submission

CHALLENGE ACTIVITY 3.13.2: Looking for characters.

Write an expression to detect that the first character of userInput matches firstLetter.


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

int main() {
   string userInput;
   char firstLetter;

   getline(cin, userInput);
   cin >> firstLetter;

   if (/* Your solution goes here  */) {
      cout << "Found match: " << firstLetter << endl;
```

```
14    }
15    else {
16       cout << "No match: " << firstLetter << endl;
17    }
```

Run

View your last submission

3.14 Character operations

Including the ***cctype library*** via `#include <cctype>` provides access to several functions for working with characters. ctype stands for character type. The first c indicates the library is originally from the C language.

Table 3.14.1: 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	`letter = toupper('a')`
						`// A`
						`letter = toupper('A')`
						`// A`
						`letter = toupper('3')`
						`// 3`

isdigit(c)	true if digit: 0-9.	`isdigit('x')`
		`// false`
		`isdigit('6')`
		`// true`
				tolower(c)	Lowercase version	`letter = tolower('A')`
						`// a`
						`letter = tolower('a')`
						`// a`
						`letter = tolower('3')`
						`// 3`

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++).

Figure 3.14.1: State abbreviation capitalization.


```
#include <iostream>
#include <cctype>
using namespace std;

int main() {
   char let0;
   char let1;

   cout << "Enter a two-letter state
abbreviation: ";
   cin >> let0;
   cin >> let1;

   if ( ! (isalpha(let0) && isalpha(let1)) ) {
      cout << "Error: Both are not letters." <<
endl;
   }
   else {
      let0 = toupper(let0);
      let1 = toupper(let1);
      cout << "Capitalized: " << let0 << let1 <<
endl;
   }

   return 0;
}
```

```
Enter a two-letter state
abbreviation: az
Capitalized: AZ
...
Enter a two-letter state
abbreviation: AZ
Capitalized: AZ
...
Enter a two-letter state
abbreviation: Mn
Capitalized: MN
...
Enter a two-letter state
abbreviation: 5x
Error: Both are not letters.
...
Enter a two-letter state
abbreviation: A@
Error: Both are not letters.
```

PARTICIPATION ACTIVITY 3.14.1: Character functions.

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

1) isalpha('7')

- True
- False

2) isalpha(userStr.at(0))

- True
- False

3) isspace(userStr.at(3))

- True

- False

4) isdigit(userStr.at(6))

- True
- False

5) toupper(userStr.at(1)) returns 'E'.

- True
- False

6) tolower(userStr.at(2)) yields an error because 'y' is already lower case.

- True
- False

7) tolower(userStr.at(6)) yields an error because '?' is not alphabetic.

- True
- False

8) After tolower(userStr.at(0)), userStr becomes "hey #1?"

- True
- False

CHALLENGE ACTIVITY 3.14.1: String with digit.

Set hasDigit to true if the 3-character passCode contains a digit.

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

int main() {
   bool hasDigit;
   string passCode;

   hasDigit = false;
   cin >> passCode;

   /* Your solution goes here  */
```

```
14    if (hasDigit) {
15       cout << "Has a digit." << endl;
16    }
17    else {
```

Run

View your last submission

CHALLENGE ACTIVITY 3.14.2: Alphabetic replace.

Replace any alphabetic character with '_' in 2-character string passCode. Ex: If passCode is "9a", output is:

```
9_
```

Hint: Use two if statements to check each of the two characters in the string, using isalpha().

```
1  #include <iostream>
2  #include <string>
3  #include <cctype>
4  using namespace std;
5
6  int main() {
7     string passCode;
8
9     cin >> passCode;
10
11    /* Your solution goes here  */
12
13    cout << passCode << endl;
14    return 0;
15 }
```

Run

View your last submission

3.15 More string operations

Finding in a string / Getting a substring

The string library provides numerous useful functions, including functions for finding a character or string within a string, or getting a substring of a string.

Table 3.15.1: find() and substr() functions, invoked as myString.find().

find()	***find***(item) returns index of first item occurrence, else returns string::npos (a constant defined in the string library). Item may be char, string variable, string literal (or char array).
	
	find(item, indx) starts at index indx.
		`// userText is "Help me!"`
		`userText.find('p')  // Returns 3`
		`userText.find('e')  // Returns 1 (first occurrence of e only)`
		`userText.find('z')  // Returns string::npos`
		`userText.find("me")  // Returns 5`
		`userText.find('e', 2) // Returns 6 (starts at index 2)`

substr()	***substr***(index, length) returns substring starting at index and having length characters.	`// userText is "http://google.com"`
		`userText.substr(0, 7)  // Returns "http://"`
		`userText.substr(13, 4)  // Returns ".com"`
		`userText.substr(userText.size() - 4, 4)  // Last 4: ".com"`

Figure 3.15.1: Example: Get username from email address.

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

int main() {
   string emailText;
   int atSymbolIndex;
   string emailUsername;

   cout << "Enter email address: ";
   cin  >> emailText;

   atSymbolIndex = emailText.find('@');
   if (atSymbolIndex == string::npos) {
      cout << "Address is missing @" << endl;
   }
   else {
      emailUsername = emailText.substr(0,
atSymbolIndex);
      cout << "Username: " << emailUsername <<
endl;
   }

   return 0;
}
```

```
Enter email address:
AbeLincoln@fakeemail.com
Username: AbeLincoln

...

Enter email address: swimming_is_fun
Address is missing @
```


PARTICIPATION ACTIVITY 3.15.1: find() and substr().

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

1) What does userText.find(',') return?

Check Show answer

2) What does userText.find("April") return?

Check Show answer

3) What does userText.substr(0, 3) return?

Check Show answer

4) What does userText.substr(userText.size() - 4, 4) return?

Check Show answer

5) In the above email username example, what was the second argument passed to substr()?

Check Show answer

Combining / Replacing

The string library has more functions for modifying strings.

Table 3.15.2: String modify functions, invoked as myString.push_back(c). Each increases/decreases string's length appropriately.

push_back()	***push_back***(c) appends character c to the end of a string.	`// userText is "Hello"`
		`userText.push_back('?'); // Now "Hello?"`
		`userText.size();  // Returns 6`

insert()	***insert***(indx, subStr) Inserts string subStr starting at index indx.	`// userText is "Goodbye"`
		`userText.insert(0, "Well "); // Now "Well Goodbye"`
		
		`// userText is "Goodbye"`
		`userText.insert(4, "---");  // Now "Good---bye"`

replace()	***replace***(indx, num, subStr) replaces characters at indices indx to indx+num-1 with a copy of subStr.	`// userText is "You have many gifts"`
		`userText.replace(9, 4, "a plethora of");`
		`// Now "You have a plethora of gifts"`

str1 + str2	Returns a new string that is a copy of str1 with str2 appended.
	
	If one of str1, str2 is a string, the other may be a character (or character array).
		`// userText is "A B"`
		`myString = userText + " C D";`
		`// myString is "A B C D"`
		`myString = myString + '!';`
		`// myString now "A B C D!"`
		`myString = myString + userText;`
		`// myString now "A B C D!A B"`

Figure 3.15.2: String modify example: Greeting.

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

int main() {
   string userName;
   string greetingText;
   int itemIndex;

   itemIndex = 0;

   cout << "Enter name: ";
   getline(cin, userName);

   // Combine strings using +
   greetingText = "Hello " + userName;

   // Append a period (could have used +)
   greetingText.push_back('.'); // '' not ""
   cout << greetingText << endl;

   // Insert Mr/Ms before user's name
   greetingText.insert(6, "Mr/Ms ");
   cout << greetingText << endl;

   // Replace occurrence of "Darn" by "@$#"
   if (greetingText.find("Darn") != string::npos) { // Found
      itemIndex = static_cast<int>(greetingText.find("Darn"));
      greetingText.replace(itemIndex, 4, "@#$");
   }
   cout << greetingText << endl;

   return 0;
}
```

```
Enter name: Julia
Hello Julia.
Hello Mr/Ms Julia.
Hello Mr/Ms Julia.

...

Enter name: Darn Rabbit
Hello Darn Rabbit.
Hello Mr/Ms Darn Rabbit.
Hello Mr/Ms @#$ Rabbit.
```

PARTICIPATION ACTIVITY 3.15.2: String modification functions.

str1 is "Main" and str2 is " Street" (note the leading space).

1) Use + to combine str1 and str2, so newStr should be "Main Street".

```
newStr = str1
                    ;
```


Check Show answer

2) Use push_back to append period to str2, so str2 should be " Street."

```
str2.                    ;
```

Check Show answer

3) Replace "ai" with "our" in str1, so str1 should be "Mourn". The first two arguments are just numbers.

```
str1.replace(
);
```

Check Show answer

Exploring further:

Numerous additional functions exist for strings.

- C++ string library

CHALLENGE ACTIVITY 3.15.1: Combining strings.

Assign secretID with firstName, a space, and lastName. Ex: If firstName is Barry and lastName is Allen, then output is:

```
Barry Allen
```

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

int main() {
   string secretID;
   string firstName;
   string lastName;

   cin >> firstName;
   cin >> lastName;

   /* Your solution goes here  */

   cout << secretID << endl;
   return 0;
}
```


Run

View your last submission

CHALLENGE ACTIVITY 3.15.2: Name song.

Modify songVerse to play "The Name Game" (see OxfordDictionaries.com), by replacing "(Name)" with userName but without the first letter.

Ex: If userName = "Kaitlin" and songVerse = "Banana-fana fo-f(Name)!", the program prints:

```
Banana-fana fo-faitlin!
```

Ex: If userName = "Kaitlin" and songVerse = "Fee fi mo-m(Name)", the program prints:

```
Fee fi mo-maitlin
```

Note: You may assume songVerse will always contain the substring "(Name)".


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

int main() {
   string userName;
   string songVerse;
```

```
   getline(cin, userName);
   userName = userName.substr(1, userName.size() - 1);  // Remove first character

   getline(cin, songVerse);

   // Modify songVerse to replace (Name) with userName without first character

   /* Your solution goes here  */

   cout << songVerse << endl;
```


Run

View your last submission

CHALLENGE ACTIVITY 3.15.3: Using find().

Print "Censored" if userInput contains the word "darn", else print userInput. End with newline.
Ex: If userInput is "That darn cat.", then output is:

```
Censored
```

Ex: If userInput is "Dang, that was scary!", then output is:

```
Dang, that was scary!
```

Note: If the submitted code has an out-of-range access, the system will stop running the code after a few seconds, and report "Program end never reached." The system doesn't print the test case that caused the reported message.

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

int main() {
   string userInput;

   getline(cin, userInput);

   /* Your solution goes here  */

   return 0;
}
```


Run

View your last submission

3.16 Conditional expressions

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

PARTICIPATION ACTIVITY 3.16.1: Conditional expression.

Animation captions:

1. An if-else statement can be written as a conditional expression.

A ***conditional expression*** has the form `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.

PARTICIPATION ACTIVITY 3.16.2: 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. ..

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

y = (                    ) ? 50
: x;
```

Check Show answer

2)

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

```
y = (x < 20)
```


Check Show answer

3)

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

Check Show answer

4)

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

Check Show answer

5)

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

Check Show answer

CHALLENGE ACTIVITY 3.16.1: Conditional expression: Print negative or positive.

Create a conditional expression that evaluates to string "negative" if userVal is less than 0, and "non-negative" otherwise. Ex: If userVal is -9, output is:

```
-9 is negative.
```

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

int main() {
   string condStr;
   int userVal;

   cin >> userVal;

   condStr = /* Your solution goes here  */;

   cout << userVal << " is " << condStr << "." << endl;

   return 0;
}
```


Run

View your last submission

CHALLENGE ACTIVITY 3.16.2: Conditional assignment.

Using a conditional expression, write a statement that increments numUsers if updateDirection is 1, otherwise decrements numUsers. Ex: if numUsers is 8 and updateDirection is 1, numUsers becomes 9; if updateDirection is 0, numUsers becomes 7.

Hint: Start with "numUsers = ...".

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

int main() {
   int numUsers;
   int updateDirection;

   cin >> numUsers;
   cin >> updateDirection;

   /* Your solution goes here  */

   cout << "New value is: " << numUsers << endl;
```

```
14
15     return 0;
16 }
```

Run

View your last submission

CHALLENGE ACTIVITY 3.16.3: Conditional expressions: Enter the output of the code.

3.17 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.

PARTICIPATION ACTIVITY 3.17.1: Floating-point comparisons.

Animation captions:

1. Floating-point numbers can't always be exactly represented in limited memory bits.
2. Thus, floats should not be compared with ==.
3. Compare floats for 'close enough'.

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.

The std::abs() function is overloaded to support floating-point and integer types. However, good practice is to use the fabs() function to make the operation clear.

PARTICIPATION ACTIVITY 3.17.2: Using == with floating-point numbers.

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

PARTICIPATION ACTIVITY 3.17.3: Floating-point comparisons.

Each comparison has a problem. Click on the problem.

1) fabs (x - y) == 0.0001

2) fabs (x - y) < 1.0

PARTICIPATION ACTIVITY 3.17.4: Floating point statements.

Complete the comparison for floating-point numbers.

1) Determine if double variable x is 98.6.

```
        (x - 98.6) < 0.0001
```

Check Show answer

2) Determine if double variables x and y are equal. Threshold is 0.0001.

```
fabs(x - y)
```

Check Show answer

3) Determine if double variable x is 1.0

```
fabs(        ) < 0.0001
```

Check Show answer

Figure 3.17.1: Example of comparing floating-point numbers for equality: Body temperature.

```
#include <iostream>
#include <cmath>
using namespace std;

int main() {
   double bodyTemp;

   cout << "Enter body temperature in Fahrenheit:
";
   cin >> bodyTemp;

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

   return 0;
}
```

```
Enter body temperature in Fahrenheit:
98.6
Temperature is exactly normal.

...

Enter body temperature in Fahrenheit:
90
Temperature is below normal.

...

Enter body temperature in Fahrenheit:
99
Temperature is above normal.
```


PARTICIPATION ACTIVITY 3.17.5: Body temperature in Fahrenheit.

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

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 manipulator can be used in an output statement. Such output formatting is discussed in another section.

Figure 3.17.2: Observing the inexact values stored in floating-point variables.

```
sampleValue1 using just cout: 0.2
sampleValue1 is
0.200000000000000111022302
sampleValue2 is
0.299999999999999888977698
sampleValue3 is
0.699999999999999955591079
sampleValue4 is 0
sampleValue5 is 0.25
```

```
#include <iostream>
#include <ios>
#include <iomanip>
using namespace std;

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


   cout << "sampleValue1 using just cout: "
        << sampleValue1 << endl;

   cout << setprecision(25)
        << "sampleValue1 is " << sampleValue1 <<
endl
        << "sampleValue2 is " << sampleValue2 <<
endl
        << "sampleValue3 is " << sampleValue3 <<
endl
        << "sampleValue4 is " << sampleValue4 <<
endl
        << "sampleValue5 is " << sampleValue5 <<
endl;

   return 0;
}
```


PARTICIPATION ACTIVITY 3.17.6: Inexact representation of floating-point values.

Enter a decimal value: _____

Sign	Exponent	Mantissa
0	0 0 0 0 0 0 0 0	1.0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

PARTICIPATION ACTIVITY 3.17.7: Representing floating-point numbers.

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

CHALLENGE ACTIVITY 3.17.1: Floating-point comparison: Print Equal or Not equal.

Write an expression that will cause the following code to print "Equal" if the value of sensorReading is "close enough" to targetValue. Otherwise, print "Not equal". Ex: If targetValue is 0.3333 and sensorReading is (1.0/3.0), output is:

```
Equal
```

```
#include <iostream>
#include <cmath>
using namespace std;

int main() {
   double targetValue;
   double sensorReading;

   cin >> targetValue;
   cin >> sensorReading;

   if (/* Your solution goes here  */) {
      cout << "Equal" << endl;
   }
   else {
      cout << "Not equal" << endl;
   }
```

Run

View your last submission

3.18 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.

PARTICIPATION ACTIVITY 3.18.1: Short circuit evaluation: Logical AND.

Animation captions:

1. The first operand evaluates to false, so the logical AND result is false regardless of the second operand. Short circuit evaluation skips evaluating the second operand.
2. If the first operand evaluates to true, the second operand is evaluated to determine the result.

Table 3.18.1: 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.

PARTICIPATION ACTIVITY 3.18.2: Determine which operands the program evaluates.

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)

3.19 C++ example: Salary calculation with branches

zyDE 3.19.1: Calculate salary: Calculate overtime using branches.

The following program calculates weekly salary and assumes work-hours-per-week limi

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

Overtime pay can be calculated with pseudocode as follows (assuming a weekly limit o
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 weeklyHours. Run the program again.

Load default templ

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

int main() {
   int hourlyWage;
   int weeklyHours;
   int weeklySalary;
   int overtimeHours;
   const int WEEKLY_LIMIT = 40;

   cout << "Enter hourly wage: " << endl;
   cin >> hourlyWage;
```

```
13    // FIXME: Get user input value for weeklyHours
14    weeklyHours = 40;
15
16
17    if (weeklyHours <  WEEKLY_LIMIT) {
```

10 42

Run

zyDE 3.19.2: Determine tax rate.

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

Load default templ

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     int annualSalary;
6     double taxRate;
7     int taxToPay;
8
9     cout << "Enter annual salary: " << endl;
10    cin >> annualSalary;
11
12    // Determine the tax rate from the annual salary
13    // FIXME: Write code to address the challenge question above
14    if (annualSalary <= 20000) {
15       taxRate = 0.10;
```

```
16    }
17    else if (annualSalary <= 50000) {
```

120000

Run

3.20 C++ example: Search for name using branches

zyDE 3.20.1: Search for name using branches.

A ***core generic top-level domain (core gTLD)*** name is one of the following Internet dom .com, .net, .org, and .info (ICANN: gTLDs). The following program asks the user to input name and prints whether that name is a gTLD. The program uses the equality operators which evaluates to true if the two compared strings are identical.

1. Run the program, noting that the .info input name is not currently recognized as a
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 leadin so .com or just com.

Load default templ


```
1  #include <iostream>
2  #include <string>
3  #include <cctype>
4  using namespace std;
5
6  int main() {
7     string inputName;
8     string searchName;
9     string coreGtld1;
10    string coreGtld2;
11    string coreGtld3;
```

```
12    // FIXME: Add a fourth core gTLD: .info
13    bool isCoreGtld = false;
14
15    coreGtld1 = ".com";
16    coreGtld2 = ".net";
17    coreGtld3 = ".org";
```

```
.info
```


Run

Below is a solution to the above problem.

zyDE 3.20.2: Search for name using branches.

A ***core generic top-level domain (core gTLD)*** name is one of the following Internet dom .com, .net, .org, and .info (ICANN: gTLDs). The following program asks the user to input name and prints whether that name is a gTLD. The program uses the equality operators which evaluates to true if the two compared strings are identical.

1. Run the program, noting that the .info input name is not currently recognized as a
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 leadin so .com or just com.

Load default templ

```
1  #include <iostream>
2  #include <string>
3  #include <cctype>
4  using namespace std;
5
6  int main() {
7     string inputName;
8     string searchName;
9     string coreGtld1;
10    string coreGtld2;
11    string coreGtld3;
12    // FIXME: Add a fourth core gTLD: .info
13    bool isCoreGtld = false;
14
15    coreGtld1 = ".com";
```

```
16    coreGtld2 = ".net";
17    coreGtld3 = ".org";
```

.info

Run

3.21 LAB: Interstate highway numbers

Primary U.S. interstate highways are numbered 1-99. Odd numbers (like the 5 or 95) go north/south, and evens (like the 10 or 90) go east/west. Auxiliary highways are numbered 100-999, excluding multiples of 100, and service the primary highway indicated by the rightmost two digits. Thus, I-405 services I-5, and I-290 services I-90.

Given a highway number, indicate whether it is a primary or auxiliary highway. If auxiliary, indicate what primary highway it serves. Also indicate if the (primary) highway runs north/south or east/west.

Ex: If the input is:

```
90
```

the output is:

```
I-90 is primary, going east/west.
```

Ex: If the input is:

```
290
```

the output is:


```
I-290 is auxiliary, serving I-90, going east/west.
```

Ex: If the input is:

```
0
```

the output is:

```
0 is not a valid interstate highway number.
```

Ex: If the input is:

```
200
```

the output is:


```
200 is not a valid interstate highway number.
```

See Wikipedia for more info on highway numbering.

LAB ACTIVITY | 3.21.1: LAB: Interstate highway numbers | 10 / 10

main.cpp

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

Develop mode | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program | Input (from above) → **main.cpp** (Your program) → Outp

Program output displayed here

Signature of your work What is this?

Retrieving signature

3.22 LAB: Leap year

A year in the modern Gregorian Calendar consists of 365 days. In reality, the earth takes longer to rotate around the sun. To account for the difference in time, every 4 years, a leap year takes place. A leap year is when a year has 366 days: An extra day, February 29th. The requirements for a given year to be a leap year are:

1) The year must be divisible by 4

2) If the year is a century year (1700, 1800, etc.), the year must be evenly divisible by 400

Some example leap years are 1600, 1712, and 2016.

Write a program that takes in a year and determines whether that year is a leap year.

Ex: If the input is:

```
1712
```

the output is:

```
1712 - leap year
```

Ex: If the input is:

```
1913
```

the output is:

```
1913 - not a leap year
```


LAB ACTIVITY | 3.22.1: LAB: Leap year 10 / 10

main.cpp

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

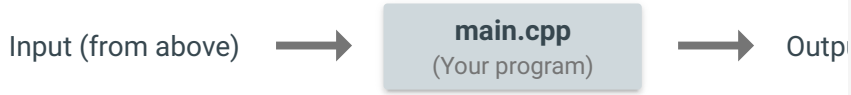
Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program



Program output displayed here

Signature of your work What is this?

Retrieving signature

3.23 LAB: Name format

Many documents use a specific format for a person's name. Write a program whose input is:

firstName middleName lastName

and whose output is:

lastName, firstInitial.middleInitial.

Ex: If the input is:

```
Pat Silly Doe
```

the output is:

```
Doe, P.S.
```


If the input has the form: firstName lastName

the output is:

lastName, firstInitial.

Ex: If the input is:

```
Julia Clark
```

the output is:

```
Clark, J.
```

LAB ACTIVITY | 3.23.1: LAB: Name format | 10 / 10

main.cpp

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

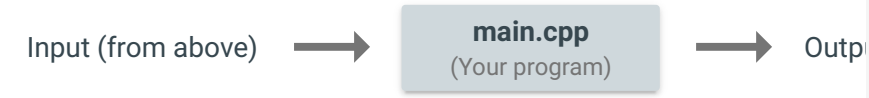

Develop mode | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program



Program output displayed here

Signature of your work What is this?

Retrieving signature