3.1 If-else

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

Construct 3.1.1: If-else statement.

```
// Statements that execute before the branches

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

// Statements that execute after the branches
```

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

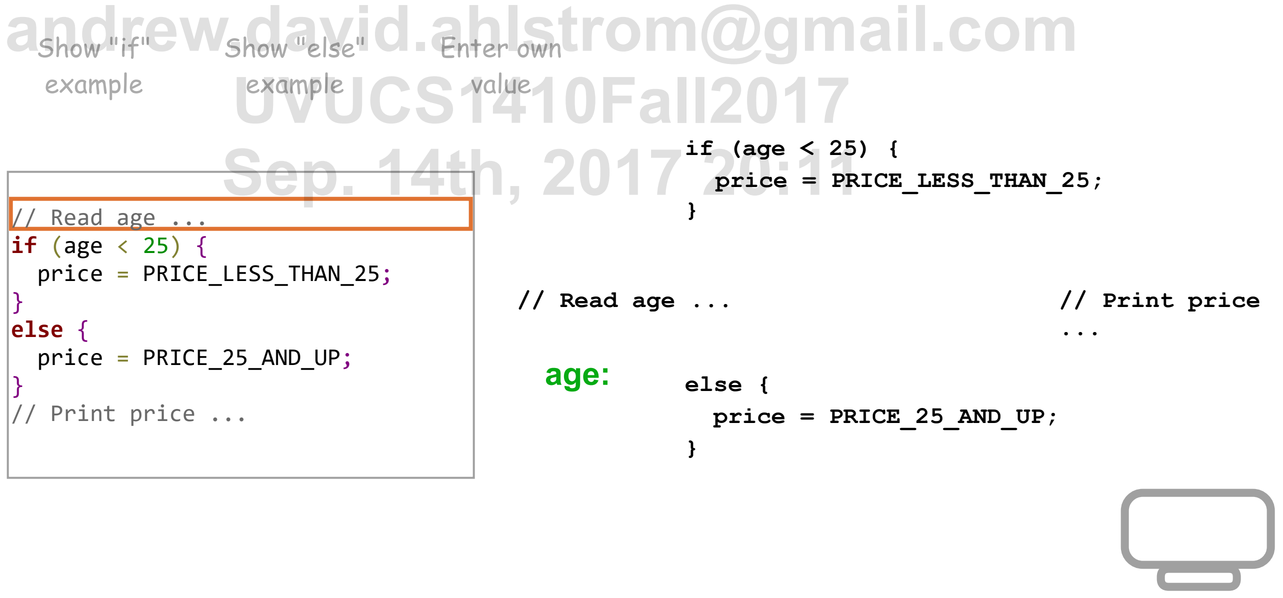
```
#include <iostream>
using namespace std;

int main() {
   const int PRICE_LESS_THAN_25 = 4800; // For ages < 25
   const int PRICE_25_AND_UP    = 2200; // For ages 25 and up
   int userAge                  = 0;    // Years
   int insurancePrice           = 0;    // Dollars

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

   if (userAge < 25) {
      insurancePrice = PRICE_LESS_THAN_25;
      cout << "(executed first branch)" << endl;
   }
   else {
      insurancePrice = PRICE_25_AND_UP;
      cout << "(executed second branch)" << endl;
   }

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

   return 0;
}
```

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

...

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

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

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

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



```
// Read age ...
if (age < 25) {
  price = PRICE_LESS_THAN_25;
}
else {
  price = PRICE_25_AND_UP;
}
// Print price ...
```

```
// Read age ...

if (age < 25) {
  price = PRICE_LESS_THAN_25;
}

else {
  price = PRICE_25_AND_UP;
}

// Print price
...
```

View animation caption(s)

PARTICIPATION ACTIVITY 3.1.2: 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 3.1.3: Writing an if-else statement.

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

1) If userAge is greater than 62, assign 15 to discount. Else, assign 0 to discount.

Check Show answer

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

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

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

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

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

   cout << "Enter an integer: ";
   cin >> userVal;

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

   cout << "The absolute value of " << userVal;
   cout << " is " << absVal << endl;

   return 0;
}
```

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

...

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

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

PARTICIPATION ACTIVITY 3.1.4: If without else.

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

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

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

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

Start

```
// Braces omitted
// but works

if (userKey == 'a')
   totalVal = 1;
else
   totalVal = 2;
```

```
// Statement added
// totalVal ALWAYS 2
// Indents irrelevant

if (userKey == 'a')
   totalVal = 1;
else
   i = i + 1;
   totalVal = 2;
```

```
// Compiler sees
// it this way

if (userKey == 'a')
   totalVal = 1;
else
   i = i + 1;
totalVal = 2;
```

```
// Always using bra
// prevents the err

if (userKey == 'a')
   totalVal = 1;
}
else {
   i = i + 1;
   totalVal = 2;
}
```

totalVal: 1

totalVal: 2

totalVal: 1

PARTICIPATION ACTIVITY 3.1.6: 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 3.1.1: Enter the output for the if-else branches.

Start

Type the program's output.

a
h

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

int main() {
   int numApples = 6;

   if (numApples < 8) {
      cout << "a" << endl;
   }
   else {
      cout << "d" << endl;
   }

   cout << "h" << endl;

   return 0;
}
```

1	2	3	4

Check Next

CHALLENGE ACTIVITY : Basic if-else expression.

Write an expression that will cause the following code to print "less than 18" if the value of userAge is less than 18.

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

int main() {
   int userAge = 0;

   userAge = 17;
   if (/* Your solution goes here  */) {
      cout << "less than 18" << endl;
   }
   else {
     cout << "18 or more" << endl;
   }

   return 0;
}
```

Run

View your last submission

CHALLENGE ACTIVITY : Basic if-else.

Write an if-else statement for the following:
If userTickets is less than 5, execute numTickets = 1. Else, execute numTickets = userTickets.
Ex: if userTickets is 3, then numTickets = 1.

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

int main() {
   int numTickets = 0;
   int userTickets = 3;

   /* Your solution goes here  */

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

Run

View your last submission

3.2 Relational and equality operators

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

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

Relational and equality operators	Description
a **<** b	a is ***less-than*** b
a **>** b	a is ***greater-than*** b

a **<=** b	a is ***less-than-or-equal-to*** b
a **>=** b	a is ***greater-than-or-equal-to*** b
a **==** b	a is ***equal to*** b
a **!=** b	a is ***not equal to*** b

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

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

Note that equality is ==, not =.

PARTICIPATION ACTIVITY 3.2.1: Expressions with relational and equality operators.

Type the operator to complete the desired expression.

```
if (expression)  {
    ...
}
else {
    ...
}
```

1) numDogs is 0

numDogs ____ 0

Check Show answer

2) numDogs is greater than 10

numDogs ____ 10

Check Show answer

3) numCars is greater than or equal to 5

numCars ____ 5

Check Show answer

4) numCars is 5 or greater

numCars [ ] 5

Check Show answer

5) numDogs and numCats are the same

numDogs [ ] numCats

Check Show answer

6) numDogs and numCats differ

numDogs [ ] numCats

Check Show answer

7) numDogs is either less-than or greater-than numCats

numDogs [ ] numCats

Check Show answer

8) centsLost is a negative number

centsLost [ ] 0

Check Show answer

9) userChar is the character 'x'.

userChar [ ] 'x'

Check Show answer

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

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

Load default template...

```

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

```
99
```

```
int main() {
   int userNum = 0;

   cout << "Enter a number: " << endl;
   cin  >> userNum;

   if (userNum == 0) {
      cout << "Zero" << endl;
   }
   else {
      cout << "Non-zero" << endl;
   }

   return 0;
}


```

Run

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

The operators 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.

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

PARTICIPATION ACTIVITY 3.2.3: 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

PARTICIPATION ACTIVITY 3.2.4: 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 == 3.25
- OK
- Not OK

PARTICIPATION ACTIVITY 3.2.5: Comparing various types (continued).

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

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

Start

Type the program's output.

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

int main() {
   int numEggs = 6;

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

   cout << "g" << endl;

   return 0;
}
```

```
e
g
```

1 2 3 4

Check Next

CHALLENGE ACTIVITY : If-else expression: Detect greater than 100.

Write an expression that will print "Dollar or more" if the value of numCents is at least a dollar (100 cents is a dollar).
Ex: If numCents is 109, output is "Dollar or more".

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

int main() {
   int numCents = 0;

   numCents = 109;

   if (/* Your solution goes here  */) {
      cout << "Dollar or more" << endl;
   }
   else {
      cout << "Not a dollar" << endl;
   }

   return 0;
}
```

Run

View your last submission

CHALLENGE ACTIVITY : If-else statement: Fix errors.

Re type the following code and fix any errors. The code should check if userNum is 2.

```
if (userNum = 2) {
   cout << "Num is two" << endl;
}
else {
   cout << "Num is not two" << endl;
}
```

(Notes)

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

int main() {
   int userNum = 0;

   userNum = 2;

   /* Your solution goes here  */

   return 0;
}
```

Run

View your last submission

CHALLENGE ACTIVITY : 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 = 0;

   patronAge = 55;

   /* Your solution goes here  */

   return 0;
}

```

Run

View your last submission

3.3 Multiple if-else branches

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

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

```
if (expr1) {

}
else if (expr2) {

}

...

else if (exprN) {

}
else {

}
```

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

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

int main() {
   int numYears = 0;

   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 3.3.1: Multi-branch if-else.

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

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

1) numSales is 2

Check Show answer

2) numSales is 0

Check Show answer

3) numSales is 7

Check Show answer

PARTICIPATION ACTIVITY 3.3.2: Complete the multi-branch if-else.

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

1) Fill in the missing line of code.

Check Show answer

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

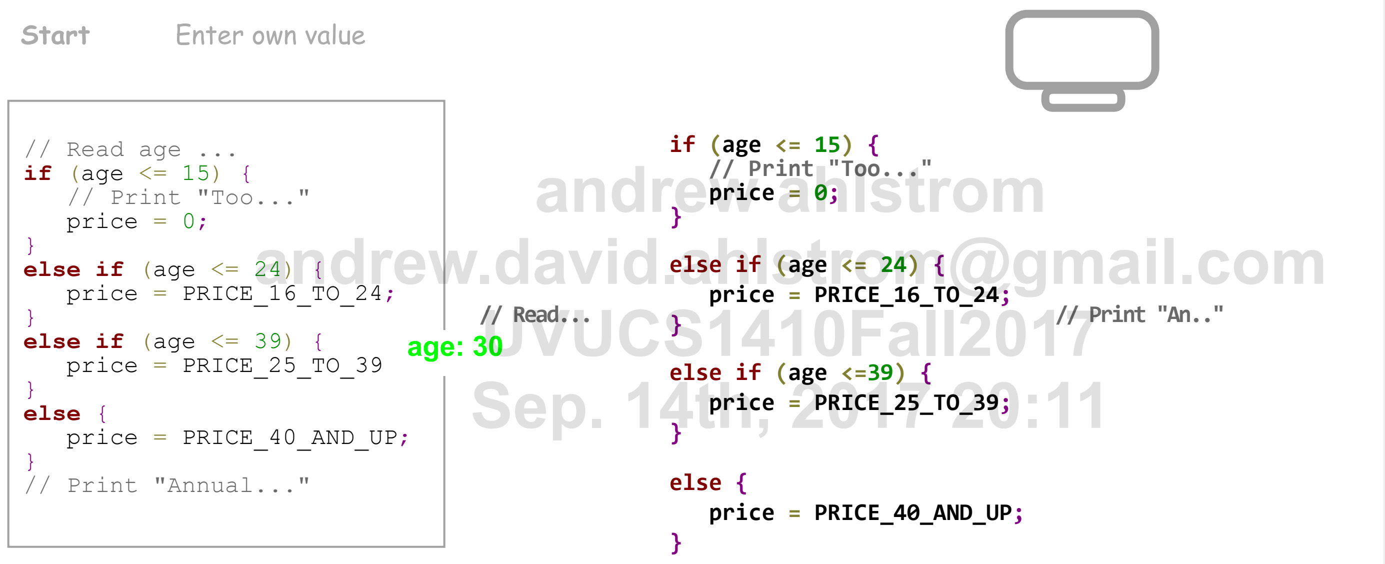
Figure 3.3.2: Using sequential nature of multi-branch if-else for ranges: Insurance prices.

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

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

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

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

   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
```

PARTICIPATION ACTIVITY 3.3.3: Only one branch will execute in a multi-branch if-else arrangement.

Start   Enter own value



```
// Read age ...
if (age <= 15) {
   // Print "Too..."
   price = 0;
}
else if (age <= 24) {
   price = PRICE_16_TO_24;
}
else if (age <= 39) {
   price = PRICE_25_TO_39
}
else {
   price = PRICE_40_AND_UP;
}
// Print "Annual..."
```

```
if (age <= 15) {
   // Print "Too..."
   price = 0;
}

else if (age <= 24) {
   price = PRICE_16_TO_24;
}

else if (age <=39) {
   price = PRICE_25_TO_39;
}

else {
   price = PRICE_40_AND_UP;
}
```

View animation caption(s)

PARTICIPATION ACTIVITY 3.3.4: Ranges and multi-branch if-else.

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

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

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 3.3.5: Complete the multi-branch code.

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

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

Figure 3.3.3: Nested if-else.

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

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

Figure 3.3.4: Multiple distinct if statements.

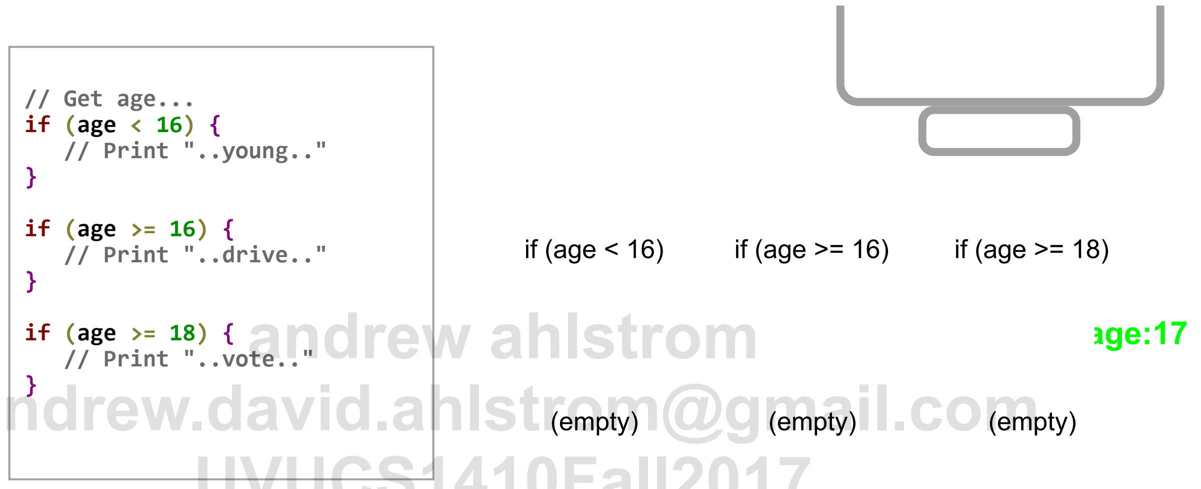
```
#include <iostream>
using namespace std;

int main() {
  int userAge = 0;

  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 >= 16) {
     cout << "You are old enough to drive." << endl;
  }

  if (userAge >= 18) {
     cout << "You are old enough to vote." << endl;
  }

  if (userAge >= 25) {
     cout << "Most car rental companies will rent to you." <<
endl;
  }

  if (userAge >= 35) {
     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 3.3.6: Multiple if statements.

Start   Enter own value

```
..drive..
```



```
// Get age...
if (age < 16) {
   // Print "..young.."
}

if (age >= 16) {
   // Print "..drive.."
}

if (age >= 18) {
   // Print "..vote.."
}
```

View animation caption(s)

PARTICIPATION ACTIVITY

3.3.7: 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 = 5;

   if (numItems > 3) {
      cout << "c" << endl;
   }
   else if (numItems <= 9) {
      cout << "d" << endl;
   }
   else {
      cout << "k" << endl;
   }

   cout << "r" << endl;

   return 0;
}
```

```
c
r
```

1 2 3 4

Check Next

CHALLENGE ACTIVITY : If-else statement: Fix errors.

Re type the code and fix any errors. The code should convert negative numbers to 0.

```
if (userNum >= 0)
   cout << "Non-negative" << endl;
else
   cout << "Negative; converting to 0" << endl;
   userNum = 0;

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

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

int main() {
   int userNum = 0;
   /* Your solution goes here  */

   return 0;
}
```

Run

View your last submission

CHALLENGE ACTIVITY : Multiple branch If-else statement: Print century.

Write an if-else statement with multiple branches. If givenYear is 2100 or greater, print "Distant future" (without quotes). Else, if givenYear is 2000 or greater (2000-2099), print "21st century". Else, if givenYear is 1900 or greater (1900-1999), print "20th century". Else (1899 or earlier), print "Long ago". Do NOT end with newline.

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

int main() {
   int givenYear = 0;

   givenYear = 1776;

   /* Your solution goes here  */

   return 0;
}
```

Run

View your last submission

CHALLENGE ACTIVITY : Multiple if statements: Print car info.

Write multiple if statements. If carYear is 1969 or earlier, print "Probably has few safety features." If 1970 or higher, print "Probably has seat belts." If 1990 or higher, print "Probably has anti-lock brakes." If 2000 or higher, print "Probably has air bags." End each phrase with period and newline. Ex: carYear = 1995 prints:

```
Probably has seat belts.
Probably has anti-lock brakes.
```

(Notes)

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

int main() {
   int carYear = 0;

   carYear = 1940;

   /* Your solution goes here  */

   return 0;
}
```

Run

View your last submission

3.4 Logical operators

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

Table 3.4.1: Logical operators.

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

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

Table 3.4.2: Logical operators examples.

Given age = 19, days = 7, userChar = 'q'	
`(age > 16) && (age < 25)`	true, because both operands are true.
`(age > 16) && (days > 10)`	false, because one operand is true and the other is false (days > 10 is false).
`(age > 16) || (days > 10)`	true, because at least one operand is true (age > 16 is true).
`!(days > 10)`	true, because operand is false.
`!(age > 16)`	false, because operand is true.
`!(userChar == 'q')`	false, because operand is true.

PARTICIPATION ACTIVITY 3.4.1: Evaluating expressions with logical operators.

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

1) `numPeople >= 10`

- true
- false

2) `(numPeople >= 10) && (numCars > 2)`

- true

○ false

3) `(numPeople >= 20) || (numCars > 1)`

○ true

○ false

4) `!(numCars < 5)`

○ true

○ false

5) `!(userKey == 'a')`

○ true

○ false

6) `userKey != 'a'`

○ true

○ false

7) `!((numPeople >= 10) && (numCars > 2))`

○ true

○ false

8) `(userKey == 'x') || ((numPeople > 5) && (numCars > 1))`

○ true

○ false

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

1) days is greater than 30 and less than 90

```
if ( (days > 30) ________ (days
< 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) && (
[                    ] ) ) {
   ...
}
```

Check Show answer

4) numDogs is 3 or more and numCats is 3 or more.

```
if ( (numDogs >= 3)
[                  ] ) {
   ...
}
```

Check Show answer

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

```
if ( [                  ] ) {
   ...
}
```

Check Show answer

6) num is a 3-digit positive integer, such as 100, 989, or 523, but not 55, 1000, or -4.

For most direct readability, your

expression should compare directly with the smallest and largest 3-digit number.

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

Check Show answer

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

1) userNum is less than -5 or greater than 10: `(userNum < -5) && (userNum > 10)`

- ○ Correct
- ○ Incorrect

2) userNum is not greater than 100: `(userNum !> 100)`

- ○ Correct
- ○ Incorrect

3) userNum is neither 5 nor 10: `!( (userNum == 5) || (userNum == 10) )`

- ○ Correct
- ○ Incorrect

4) userNum is between 10 and 20, inclusive `( (userNum >= 10) || (userNum <= 20) )`

- ○ Correct
- ○ Incorrect

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

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

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

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

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

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

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

Table 3.4.3: Precedence rules for logical and relational operators.

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

		because & has precedence over |.
|	Then, the bitwise OR operator is evaluated.	`x == 5 | y == 10 && z != 10` is evaluated as `((x == 5) | (y == 10)) && (z != 10))` because | has precedence over &&.
&&	Then, the logical AND operator is evaluated.	`x == 5 || y == 10 && z != 10` is evaluated as `(x == 5) || ((y == 10) && (z != 10))` because && has precedence over ||.
||	Finally, the logical OR operator is evaluated.	

PARTICIPATION ACTIVITY 3.4.4: Logical expression simulator.

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

```
int x = 7 ;
int y = 5 ;
if (                ) {
   ...
}
```

Run code

Output is:

```
Awaiting your input...
```

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

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

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

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

PARTICIPATION ACTIVITY 3.4.5: Order of evaluation.

Which of the following expressions illustrate the correct order of evaluation with 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)

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

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

PARTICIPATION ACTIVITY 3.4.6: Mixing bitwise and logical operators.

1) `x == 3 | y > 1 && z != 3`

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

- ○ (x == 3) | ((y > 1) && (z != 3))
- ○ ((x == 3) | (y > 1)) && (z != 3)

2) `x == 3 & y > 1 || z != 3`

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

- ○ ((x == 3) & (y > 1)) || (z != 3)
- ○ (x == 3) & ((y > 1) || (z != 3))

3) `x < 7 | y >= 10 && z == 15`

For which values of x, y, and z does the expression evaluate to true?

- ○ x = 4, y = 11, and z = 10
- ○ x = 4, y = 11, and z = 15

CHALLENGE ACTIVITY : Detect specific values.

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

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

int main() {
   int specialNum = 0;

   specialNum = 17;

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

   return 0;
}
```

Run

View your last submission

CHALLENGE ACTIVITY : 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 = 0;

   userAge = 17;

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

   return 0;
}
```

Run

View your last submission

3.5 Switch statements

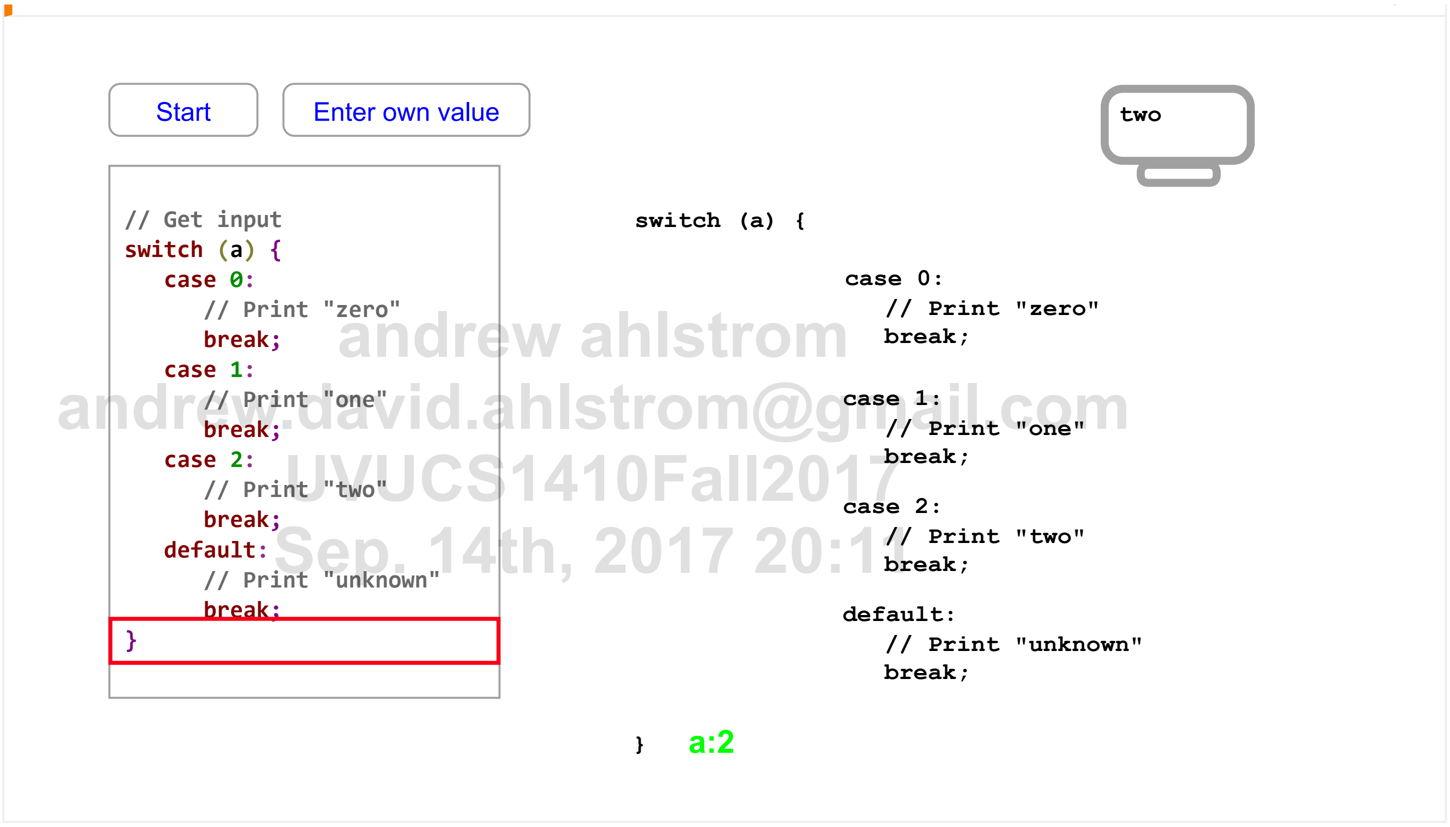
A ***switch*** statement can more clearly represent multi-branch behavior involving a variable being compared to constant values. The program executes the first ***case*** whose constant expression matches the value of the switch expression, executes that case's statements, and then jumps to the end.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

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

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

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

int main() {
   int dogAgeYears = 0;

   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;
}
```

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

...

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

PARTICIPATION ACTIVITY 3.5.1: Switch statement.



```
// Get input
switch (a) {
   case 0:
      // Print "zero"
      break;
   case 1:
      // Print "one"
      break;
   case 2:
      // Print "two"
      break;
   default:
      // Print "unknown"
      break;
}
```

```
switch (a) {

      case 0:
         // Print "zero"
         break;

      case 1:
         // Print "one"
         break;

      case 2:
         // Print "two"
         break;

      default:
         // Print "unknown"
         break;

}   a:2
```

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

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

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

PARTICIPATION ACTIVITY 3.5.2: Switch statement.

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

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

   case 3:
      numItems = 12;
      break;

   case 4:
      numItems = 99;
      break;

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

1) userVal = 3;

Check Show answer

2) userVal = 0;

Check Show answer

3) userVal = 2;

Check Show answer

Construct 3.5.1: Switch statement general form.

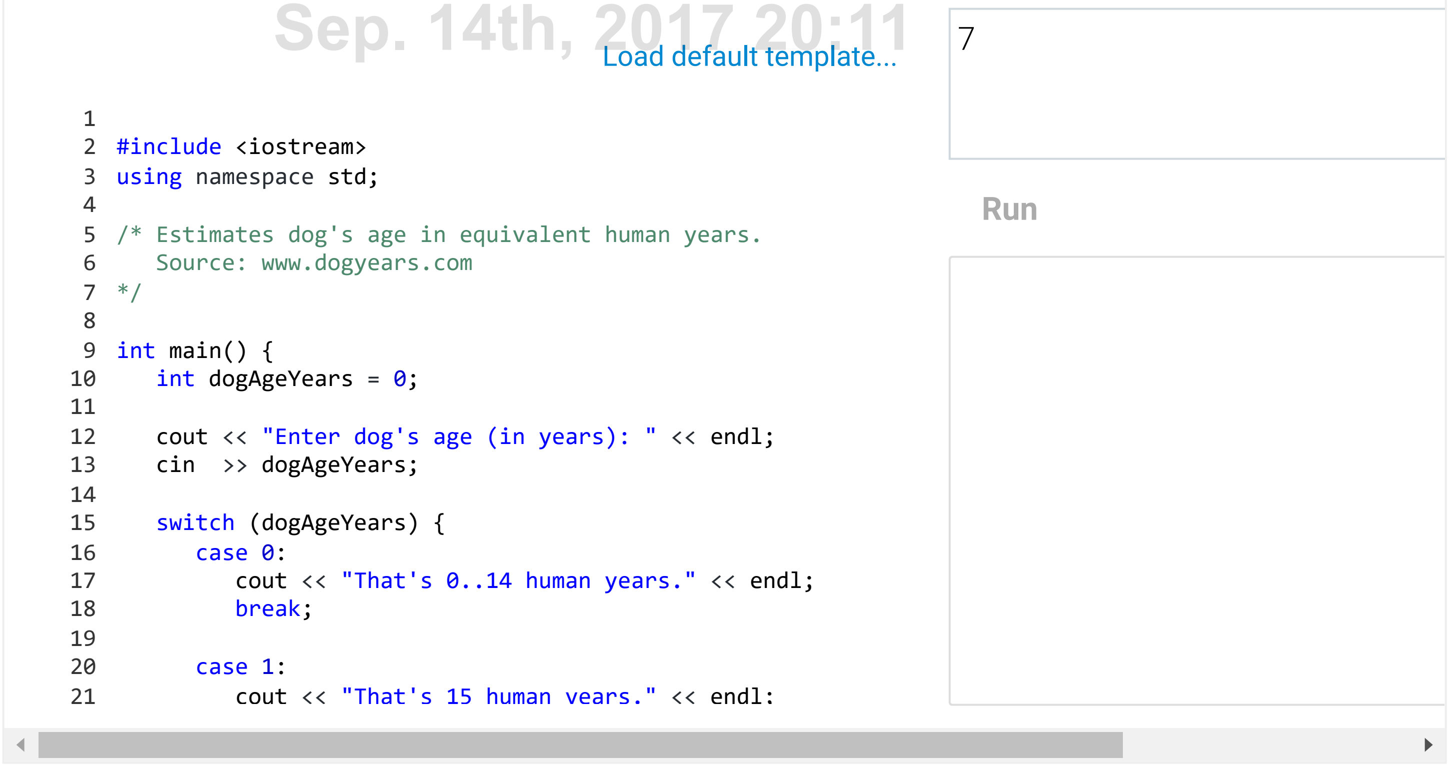
```
switch (expression) {
   case constantExpr1:
      // Statements
      break;

   case constantExpr2:
      // Statements
      break;

   ...

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

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

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

PARTICIPATION ACTIVITY 3.5.3: Switch statement: Numbers to words.

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



```

#include <iostream>
using namespace std;

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

int main() {
   int dogAgeYears = 0;

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

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

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

```
7
```

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.

Figure 3.5.3: Switch example: Dog years with months.

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

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

   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;
}
```

```
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.
```

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

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

PARTICIPATION ACTIVITY 3.5.4: Switch statement.

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

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

   case 'B':
      encodedVal = 2;
      break;

   case 'C':

   case 'D':
      encodedVal = 4;
      break;

   case 'E':
      encodedVal = 5;

   case 'F':
      encodedVal = 6;
      break;

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

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 : 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. Do not get input from the user; nextChoice is assigned in main().

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

int main() {
   int nextChoice = 0;

   nextChoice = 2;

   /* Your solution goes here  */

   return 0;
}
```

Run

View your last submission

CHALLENGE ACTIVITY : 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 = '?';

   origLetter = 'a';

   /* Your solution goes here  */

   return 0;
}
```

Run

View your last submission

3.6 Boolean data types

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

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

Figure 3.6.1: Example using variables of bool data type.

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

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

   cout << "Enter any integer: ";
   cin  >> userNum;

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

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

   cout << "(isLarge: " << isLarge;
   cout << " isNeg: " << isNeg << ")" << endl;

   cout << "You entered a ";
   if (isLarge && isNeg) {
      cout << "large negative number." << endl;
   }
   else if (isLarge && !isNeg) {
      cout << "large positive number." << endl;
   }
   else {
      cout << "small number." << endl;
   }

   return 0;
}
```

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

...

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

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

PARTICIPATION ACTIVITY 3.6.1: Boolean variables.

1) Write a statement to declare and initialize a Boolean variable named `night` to false.

Check Show answer

2) What is stored in variable `isFamous` after executing the following statements, `true` or `false`?

```
bool isTall = false;
bool isRich = true;
bool isFamous = false;
if (isTall && isRich)
{
   isFamous = true;
}
```

Check Show answer

CHALLENGE ACTIVITY : Using bool.

Write code to assign true to isTeenager if kidAge is 13 to 19 inclusive.

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

int main() {
   bool isTeenager = false;
   int kidAge = 0;

   kidAge = 13;

   /* Your solution goes here  */

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

   return 0;
}
```

Run

View your last submission

CHALLENGE ACTIVITY : 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 = false;
   bool isBalloon = false;

   /* Your solution goes here  */

   return 0;
}
```

Run

View your last submission

3.7 String comparisons

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

PARTICIPATION ACTIVITY 3.7.1: 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 !=.

PARTICIPATION ACTIVITY 3.7.2: 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

Figure 3.7.1: 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
```

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

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

PARTICIPATION ACTIVITY 3.7.3: Relational string comparison.

1) Complete the code by comparing string variables myName and yourName. Start with myName.

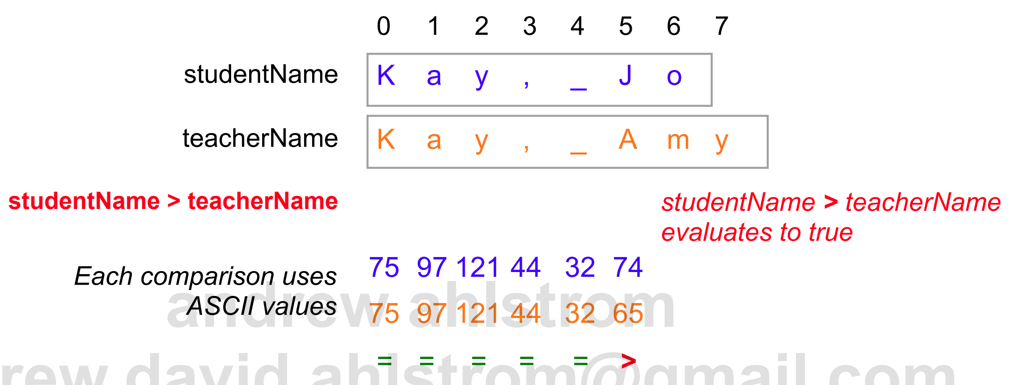
```
if (                    ) {
   cout << myName << " is
greater.";
}
```

Check Show answer

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

PARTICIPATION ACTIVITY 3.7.4: String comparison.

Start



PARTICIPATION ACTIVITY 3.7.5: 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 can compare strings while ignoring case by first converting both strings to lowercase before comparing (discussed elsewhere).

CHALLENGE ACTIVITY : String comparison: Detect word.

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

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

int main() {
   string userString;

   userString = "Quit";

   /* Your solution goes here  */

   return 0;
}
```

Run

View your last submission

CHALLENGE ACTIVITY : Print two strings in alphabetical order.

Print the two strings 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;

   firstString = "rabbits";
   secondString = "capes";

   /* Your solution goes here  */

   return 0;
}
```

Run

View your last submission

3.8 String access operations

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

at(): The notation someString.at(0) accesses the character at a particular index of a string, in this case index 0.

Figure 3.8.1: String character access: Word scramble.

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

int main() {
   string word1;
   string word2;

   cout << "Enter word (>= 5 letters): ";
   cin  >> word1;

   word2 = word1;

   cout << "Size: " << word1.size() << endl;
   // Note: Error if word1 has < 5 letters

   word2.at(0) = word1.at(3);
   word2.at(1) = word1.at(4);
   word2.at(2) = word1.at(1);
   word2.at(3) = word1.at(0);
   word2.at(4) = word1.at(2);

   cout << "Original:  " << word1 << endl;
   cout << "Scrambled: " << word2 << endl;

   return 0;
}
```

```
Enter word (>= 5 letters): forest
Size: 6
Original:  forest
Scrambled: esofrt
```

PARTICIPATION ACTIVITY

3.8.1: String access.

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

1) How many numbers do you see:
0 1 2 3

Check Show answer

2) What character is at index 1 of userText?

Check Show answer

3) What is the index of the last character, 'k', in userText?

Check Show answer

4) To what character does this evaluate:
userText.at(3)

Check Show answer

5) What is userText after the following:
userText.at(0) = 't';

Check Show answer

The string library provides useful functions for accessing information about a string.

Table 3.8.1: String info functions, invoked as someString.length().

length()	Number of characters	

	size() is the same	`// userText is "Help me!"`
		`userText.length()  // Returns 8`
		`userText.size()    // Returns 8`
		`// userText is ""`
		`userText.length()  // Returns 0`

empty()	true if length is 0	`// userText is "Help me!"`
		`userText.empty()   // Returns false`
		`// userText is ""`
		`userText.empty()   // Returns true`

find(item)	Returns index of first item occurrence, else returns string::npos, which is a constant defined in the string library. Item may be char, string variable, string literal (or char array, if you've studied that) ***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(indx, len)	Returns substring starting at index and having len	`// userText is "http://google.com"`
		`userText.substr(0, 7)                   // Returns "http://"`
		`userText.substr(13, 4)                  // Returns ".com"`
		`userText.substr(userText.length() - 4, 4) // Last 4: ".com"`

PARTICIPATION ACTIVITY 3.8.2: String access operations.

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

1) What does userText.length() return?

Check Show answer

2) What does userText.empty() return?

Check Show answer

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

Check Show answer

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

Check Show answer

5) What character does userText.at(userText.length() - 1) return?

Check Show answer

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

Check Show answer

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

Check Show answer

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

PARTICIPATION ACTIVITY 3.8.3: String access.

Animation captions:

1.

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.

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

PARTICIPATION ACTIVITY 3.8.4: Out-of-range string access.

Given userText = "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.length()) yields 'y'.

- True

○ False

CHALLENGE ACTIVITY : String library functions.

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

```
Size of userInput: 5
```

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

int main() {
   string userInput;
   int stringSize = 0;

   userInput = "Hello";

   /* Your solution goes here  */

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

   return 0;
}
```

Run

View your last submission

CHALLENGE ACTIVITY : 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 = '-';

   userInput = "banana";
   firstLetter = 'b';
```

```
    if (/* Your solution goes here  */) {
       cout << "Found match: " << firstLetter << endl;
    }
    else {
       cout << "No match: " << firstLetter << endl;
    }

    return 0;
 }

```

Run

View your last submission

CHALLENGE ACTIVITY : Using find().

Print "Censored" if userInput contains the word "darn", else print userInput. End with newline.

Note: These activities may test code with different test values. This activity will perform three tests, with userInput of "That darn cat.", then with "Dang, that was scary!", then with "I'm darning your socks.". See How to Use zyBooks.

Also 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;

   userInput = "That darn cat.";

   /* Your solution goes here  */

   return 0;
}
```

Run

View your last submission

3.9 String modify operations

The string library has several functions for modifying strings.

Table 3.9.1: String modify functions, invoked as someString.clear(). Each increases/decreases string's length appropriately.

push_back(newChar)	Appends newChar to the end.	`// userText is "Hello"`
		`userText.push_back('?'); // Now "Hello?"`
		`userText.length();       // Returns 6`

append(moreString)	Appends a copy of string moreString.	`// userText is "Hi"`
		`userText.append(" friend"); // Now "Hi friend"`
		`userText.length();          // Returns 9`

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(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"`

clear()	Deletes characters, sets size to 0.	`// userText is "Help me!"`
		`userText.clear(); // Clears string`
		`userText.size();  // Returns 0`
		`userText.at(0);   // Generates exception`

resize(num)	Resize string to have num	`// userText is "Help me!"`
		`userText.resize(4); // Now "Help"`
		`userText.size();    // Returns 4`

	characters. If decrease, drops extra chars. If increase, sets new chars to null ('\0', ASCII value 0).	

str1 + str2	Returns a new string that is a copy of str1 with str2 appended. If only one of str1 or str2 is a string variable, the other may be a character array or a character.	`// 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 3.9.1: String modify example: Greeting.

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

...

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

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

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

   // Remove the ending period
   greetingText.resize(greetingText.size() - 1);
   cout << greetingText << endl;

   // Replace occurrence of "Darn" by "@$#"
   itemIndex = greetingText.find("Darn");
   if (itemIndex >= 0) { // Found
      greetingText.replace(itemIndex, 4, "@#$");
   }
   cout << greetingText << endl;

   return 0;
}
```

PARTICIPATION ACTIVITY 3.9.1: String modification functions.

str1 is "Main", str2 is " Street" and str3 is "Western"

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" by "our" in str1, so str1 should be "Mourn". The first two arguments are just numbers.

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

Check Show answer

4) Keep only the first four chars of str3, so str3 should be "West". Type a single number.

```
str3.resize(
);
```

Check Show answer

CHALLENGE ACTIVITY : Combining strings.

Retype and correct the code provided to combine two strings separated by a space. Hint: What type of parameter does push_back expect?

```
secretID.push_back(spaceChar);
secretID.push_back(lastName);
```

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

int main() {
   string secretID = "Barry";
   string lastName = "Allen";
   char spaceChar = ' ';

   /* Your solution goes here  */

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

Run

View your last submission ⌄

CHALLENGE ACTIVITY : Name song.

Modify secondVerse to play "The Name Game" (a.k.a. "The Banana Song", see Wikipedia.org), by replacing "(Name)" with userName but without the first letter. Ex: if userName = "Katie", the program prints:

```
Banana-fana fo-fatie!
```

Note: The song verse may change, such as: Banana-fana fo-f(Name)!!!

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

int main() {
   string secondVerse = "Banana-fana fo-f(Name)!";
   string userName = "Katie";

   userName.erase(userName.begin()); // Removes first char from userName

   /* Your solution goes here  */

   cout << secondVerse << endl;

   return 0;
}
```

Run

View your last submission ⌄

3.10 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 3.10.1: Character functions return values.

isalpha(c)	true if alphabetic: a-z or A-Z	`isalpha('x') // true`
		`isalpha('6') // false`
		`isalpha('!') // false`
			toupper(c)	Uppercase version	`toupper('a') // A`
					`toupper('A') // A`
					`toupper('3') // 3`

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

PARTICIPATION ACTIVITY 3.10.1: 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 : 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 = false;
   string passCode;
   int valid = 0;

   passCode = "abc";

   /* Your solution goes here  */

   if (hasDigit) {
      cout << "Has a digit." << endl;
   }
   else {
      cout << "Has no digit." << endl;
20    }
```

Run

View your last submission

CHALLENGE ACTIVITY : Whitespace replace.

Replace any space ' ' by '_' in 2-character string passCode. Sample output for the given program:

```
1_
```

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

int main() {
   string passCode;

   passCode = "1 ";

   /* Your solution goes here  */

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

Run

View your last submission

3.11 Conditional expressions

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

PARTICIPATION ACTIVITY 3.11.1: Conditional expression.

Animation captions:

1. This if-else form can be written as a conditional expression.

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

Construct 3.11.1: Conditional expression.

```
condition ? exprWhenTrue : exprWhenFalse
```

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

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

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

PARTICIPATION ACTIVITY 3.11.2: 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 : 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. Example output when userVal = -9 for the below sample program:

```
-9 is negative.
```

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

int main() {
   string condStr;
   int userVal = 0;

   userVal = -9;

   condStr = /* Your solution goes here  */;

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

   return 0;
}
```

Run

View your last submission

CHALLENGE ACTIVITY : 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 = 0;
   int updateDirection = 0;

   numUsers = 8;
   updateDirection = 1;

   /* Your solution goes here  */

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

   return 0;
}
```

Run

View your last submission

3.12 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 3.12.1: 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 3.12.2: 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 3.12.3: 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 3.12.4: 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 3.12.1: Example of comparing floating-point numbers for equality: Body temperature.

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

int main() {
   double bodyTemp = 0.0;

   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 3.12.5: 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 3.12.2: Observing the inexact values stored in floating-point variables.

```
#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;
}
```

```
sampleValue1 using just cout: 0.2
sampleValue1 is 0.2000000000000000111022302
sampleValue2 is 0.2999999999999999888977698
sampleValue3 is 0.6999999999999999555910790
sampleValue4 is 0
sampleValue5 is 0.25
```

PARTICIPATION ACTIVITY 3.12.6: Inexact representation of floating-point values.

Enter a decimal value:

Convert

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

PARTICIPATION ACTIVITY 3.12.7: 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 : 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".

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

int main() {
   double targetValue = 0.3333;
   double sensorReading = 0.0;

   sensorReading = 1.0/3.0;

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

   return 0;
}
```

Run

View your last submission

3.13 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 3.13.1: 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 3.13.1: 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 3.13.2: 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)

3.14 C++ example: Salary calculation with branches

PARTICIPATION ACTIVITY 3.14.1: Calculate salary: Calculate overtime using branches.

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

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

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

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

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

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

int main() {
   int hourlyWage    = 0;
   int weeklyHours   = 0;
   int weeklySalary  = 0;
   int overtimeHours = 0;
   const int WEEKLY_LIMIT = 40;

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

   // FIXME: Get user input value for weeklyHours
   weeklyHours = 40;


   if (weeklyHours <= WEEKLY_LIMIT) {
      weeklySalary = weeklyHours * hourlyWage;
   }
```

```
10 42
```

Run

PARTICIPATION ACTIVITY 3.14.2: Determine tax rate.

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

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

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

int main() {
   int annualSalary = 0;
   double taxRate = 0.0;
   int taxToPay = 0;

   cout << "Enter annual salary: " << endl;
   cin >> annualSalary;

   // Determine the tax rate from the annual salary
   // FIXME: Write code to address the challenge question above
   if (annualSalary <= 20000) {
      taxRate = 0.10;
   }
   else if (annualSalary <= 50000) {
      taxRate = 0.20;
   }
   // FIXME: Add tier
```

```
120000
```

Run

3.15 C++ example: Search for name using branches

PARTICIPATION ACTIVITY 3.15.1: Search for name using branches.

A ***core generic top-level domain (core gTLD)*** name is one of the following Internet domains: .com, .net, .org, and .info (Wikipedia: gTLDs). The following program asks the user to input a name and prints whether that name is a gTLD. The program uses the 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 gTLD.
2. Extend the if-else statement to detect the .info domain name as a gTLD. Run the program again.
3. Extend the program to allow the user to enter the name with or without the leading dot, so .com or just com.

```
#include <iostream>
#include <string>
#include <cctype>
```

```
using namespace std;

int main() {
   string inputName = "";
   string searchName = "";
   string coreGtld1 = ".com";
   string coreGtld2 = ".net";
   string coreGtld3 = ".org";
   // FIXME: Add a fourth core gTLD: .info
   bool isCoreGtld = false;

   cout << endl << "Enter a top-level domain name: " << endl;
   cin >> inputName;

   // Case is irrelevant, so make all comparisons with lower case
   searchName = inputName;

   // FIXME: Allow the user to enter a name with or without a leading period
```

```
.info
```

Run

Below is a solution to the above problem.

PARTICIPATION ACTIVITY 3.15.2: Search for name using branches (solution).

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

int main() {
   string inputName = "";
   string searchName = "";
   string coreGtld1 = ".com";
   string coreGtld2 = ".net";
   string coreGtld3 = ".org";
   string coreGtld4 = ".info";
   bool isCoreGtld = false;

   cout << endl << "Enter a top-level domain name: " << endl;
   cin >> inputName;
   searchName = inputName;

   // If the user entered a name without a leading period, add one
   if ((searchName.length() > 0) && (searchName.at(0) != '.')) {
      searchName = "." + inputName;
```

```
.info
```

Run