# 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 content:

Static figure:
Pictured is a silhouette of a person named "Host". To the side of the host is the text "Party of:". To the right of the host is an abstract depiction of a restaurant layout. There are a long countertop labeled "If party of 1: ", three small tables labeled "Else If party of 2:", and three large tables labeled "Else:".

Step 1: A restaurant host seats patrons. The host seats a party of 1 at the counter. Still visible are the abstract depiction of the restaurant layout, the host, and the text "Party of:", which is next to the host. A number "1" appears, such that the text next to the host reads "Party of: 1". The number "1" then moves onto the long rectangular counter.

Step 2: A party of 2 is seated at a small table. Other-sized parties are seated at a large table. A number "2" appears before the host and moves to a small table. A number "4" appears before the host and then moves to a large table. A number "1" appears before the host and moves to the counter. A number "3" appears before the host and then moves to a large table.

Step 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. The text "If party of 1:" appears next to the counter; the text "Else If party of 2:" appears next to the small tables; and the text "Else:" appears next to the large tables.



### 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-sized 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

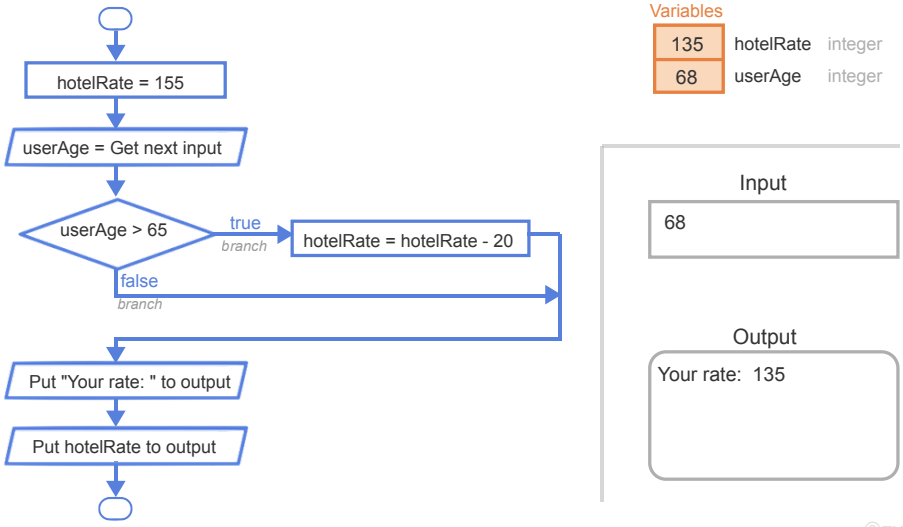
3) A party of 5 is sat _____ .
   - at a large table
   - nowhere



## Branch basics (If)

In a program, a ***branch*** is a sequence of statements only executed under a certain condition. Ex: A hotel may discount a price only for people over age 65. An ***if*** branch is a branch taken only IF an expression is true.

**PARTICIPATION ACTIVITY** 3.1.3: Branches: Hotel rate example.






### Animation content:

The animation executes the following Coral program:

```
integer hotelRate
integer userAge

hotelRate = 155
```

```
userAge = Get next input
if userAge > 65
   hotelRate = hotelRate - 20
Put "Your rate: " to output
Put hotelRate to output
```

Variables in memory is as follows:
135 hotelRate: integer
68 userAge: integer

Input is as follows:
68

Output (screen) is as follows:
Your rate: 135



### Animation captions:

1. A decision leads to two program branches. If the expression is true, the first branch executes. Else, the second branch executes.
2. If userAge is 68, then 68 > 65 is true. So the first branch executes, which discounts hotelRate.
3. Execution rejoins the other branch, and continues with subsequent statements, outputting 135. If userAge were instead 50, the output would be 155.

**PARTICIPATION ACTIVITY** 3.1.4: Branches.

Consider the hotel rate example above.

1) If userAge is 20, does the true or false branch execute?
   - ○ True branch
   - ○ False branch

2) If userAge is 20, does the executed branch update hotelRate?
   - ○ Yes
   - ○ No

3) If userAge is 20, what hotel rate does the program output?
   - ○ 155
   - ○ 135

4) If userAge is 70, what hotel rate does the program output?
   - ○ 155
   - ○ 135

5) Do the last two statements always execute for any value of userAge?

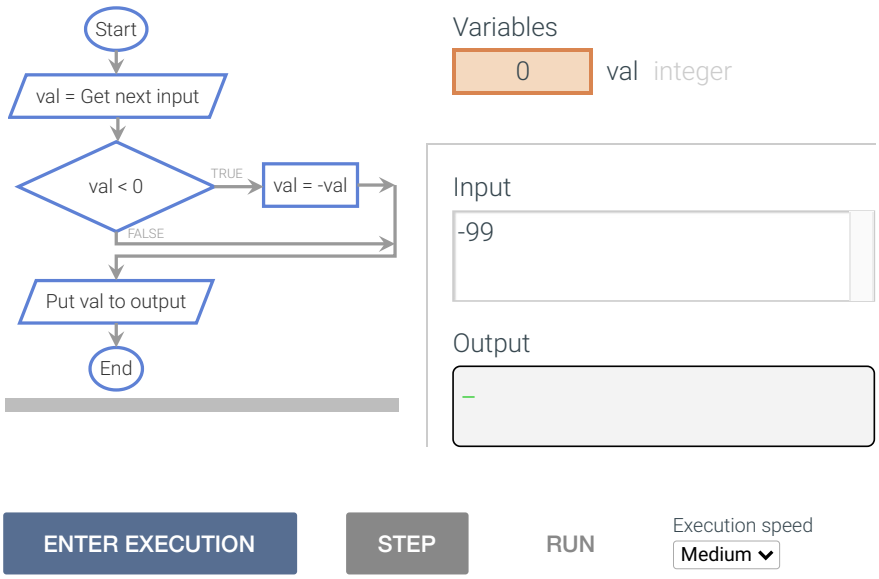
- Yes
- No

## If branch example: Absolute value

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



**PARTICIPATION ACTIVITY** 3.1.5: Computing absolute value. Full screen




**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

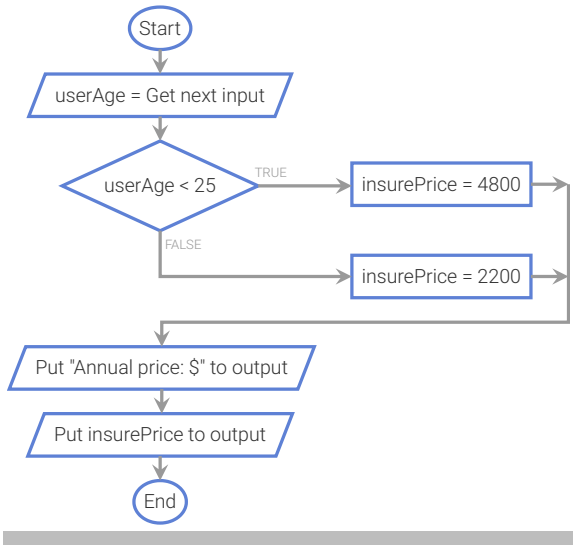


## If-else branches

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

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

**PARTICIPATION ACTIVITY** 3.1.7: Insurance price. Full screen




Variables

| | | |
|---|---|---|
| 0 | userAge | integer |
| 0 | insurePrice | integer |

Input

22

Output

–

ENTER EXECUTION STEP RUN Execution speed Medium



## 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** 3.1.8: If-else branches.

Consider the insurance price example above.

1) If userAge is 18, what price is output?
   - ○ 4800
   - ○ 2200

2) If userAge is 30, what price is output?
   - ○ 4800
   - ○ 2200

3) If userAge is 25, what price is output?
   - ○ 4800
   - ○ 2200

4) For what value of userAge will both branches execute?

- 15
- 25
- None

5) For what value of userAge will neither branch execute?

- 30
- 25
- None

6) For what value of userAge will the output statements not execute?

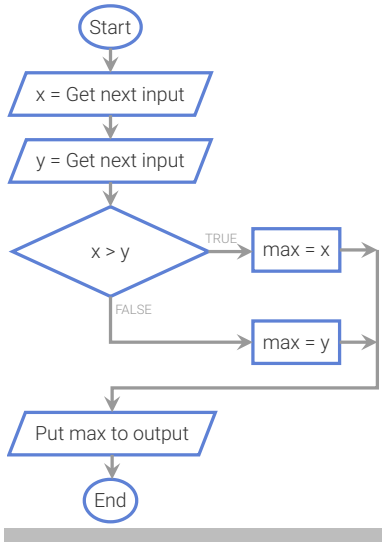
- 20
- 25
- None



## If-else example: Max

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

**PARTICIPATION ACTIVITY** 3.1.9: If-else branches example: Max. Full screen




Variables

| | | |
|---|---|---|
| 0 | x | integer |
| 0 | y | integer |
| 0 | max | integer |

Input

55 79

Output

ENTER EXECUTION STEP RUN Execution speed Medium



**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?

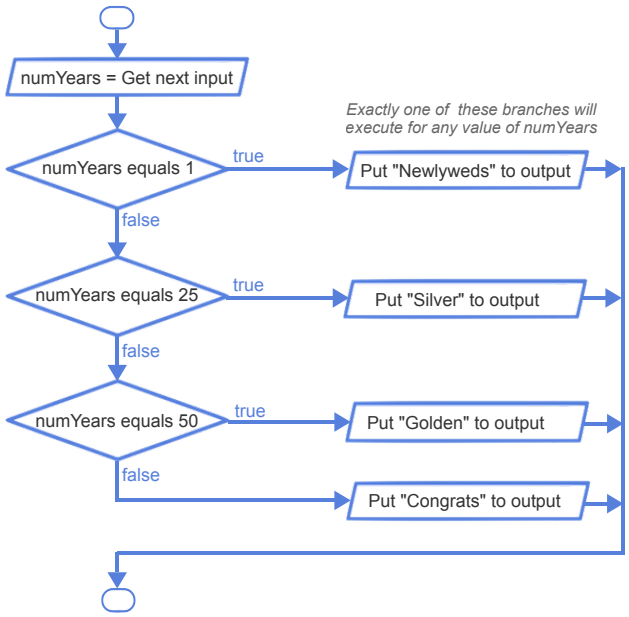
- x
- y

## 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.

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

**PARTICIPATION ACTIVITY** 3.1.11: If-elseif example: Anniversaries.




Variables

0 numYears integer

Static figure:
Pictured is a flowchart that shows the execution of a program that determines what output to display based on the value of numYears. The entry vertex leads to a vertex reading "numYears = Get next input". This leads to the next vertex, which reads "numYears equals 1" and has two possible edges to follow, one for true and one for false: the edge for true leads to a vertex reading 'Put "Newlyweds" to output'; the edge for false leads to another vertex that reads "numYears equals 25". This vertex also has both a true and a false edge: the true edge leads to a vertex reading 'Put "Silver" to output'; the false edge leads to a vertex reading "numYears equals 50". This vertex again has both a true and a false edge: the true edge leads to a vertex reading 'Put "Golden" to output'; the false edge leads to a vertex reading 'Put "Congrats" to output'.

There is a box labeled "Variables", to the side of which is the text "numYears integer". Inside the box is the number 0.

Step 1: This program detects the specific value of a variable. If numYears is 1, the first branch executes and "Newlyweds" is output. Two vertices in the flowchart are highlighted: the one that reads "numYears equals 1" and the one reading 'Put "Newlyweds" to output', the first connecting to the second by a "true" edge. There is some text above the latter highlighted vertex: "Exactly one of these branches will execute for any value of numYears".

Step 2: Else, if numYears is 25, the second branch executes and "Silver" is output. Else, if numYears is 50, the third branch executes and "Golden" is output.

Step 3: Else, the last branch executes, i.e. 'Put "Congrats" to output' executes.

### Animation captions:

1. This program detects the specific value of a variable. If numYears is 1, the first branch executes and "Newlyweds" is output.
2. Else, if numYears is 25, the second branch executes and "Silver" is output. Else, if numYears is 50, the third branch executes and "Golden" is output.
3. Else, the last 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.



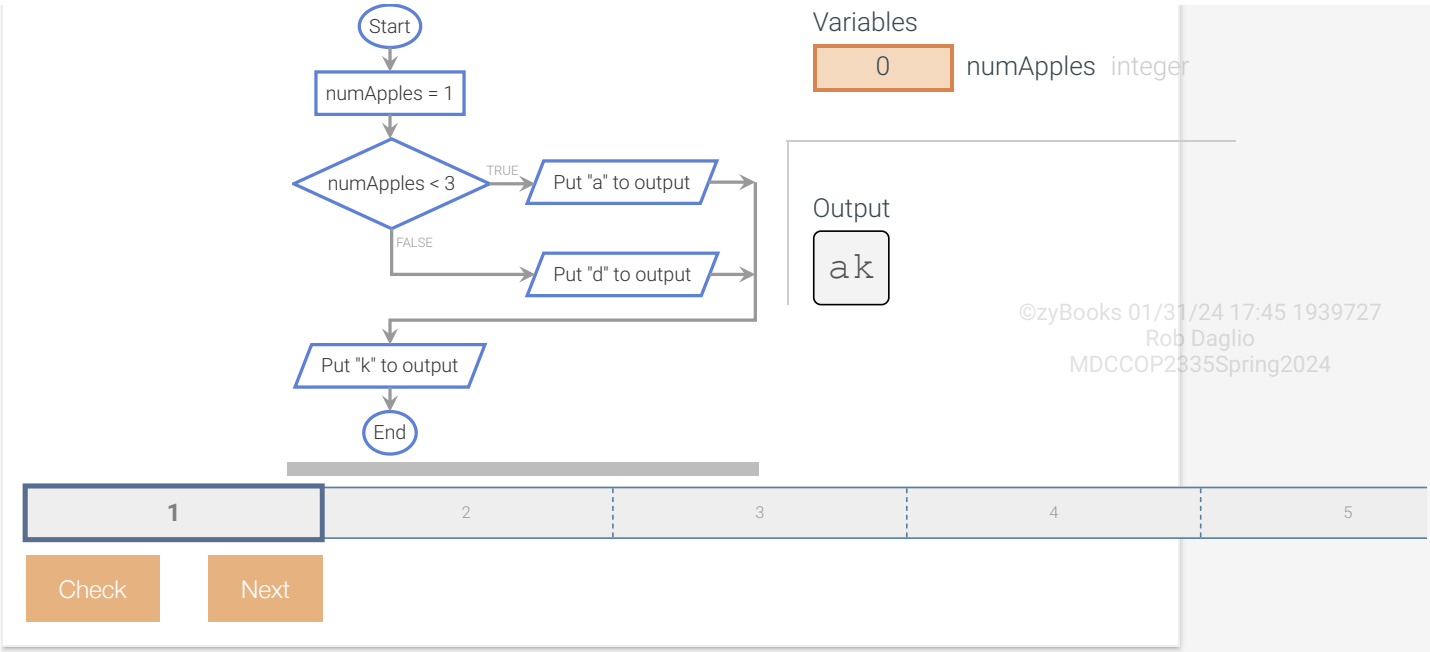
**CHALLENGE ACTIVITY** 3.1.1: If-else branches.

Note: Level 5 uses ==, known as the equality operator, to mean equals. Ex: `x == 9` means `x equals 9`.

539740.3879454.qx3zqy7

Start

Type the program's output






# 3.2 Detecting equal values with branches

## Detecting if two items are equal using an if statement

A program commonly needs to determine if two items are equal. Ex: If a hotel gives a discount for guests on their 50th wedding anniversary, a program to calculate the discount can check if a variable numYears is equal to the value 50. A programmer can use an if statement to check if two values are equal.

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.

The example below uses ==. The ***equality operator*** (***==***) evaluates to true if the left and right sides are equal. Ex: If numYears is 50, then numYears == 50 evaluates to true. Note the equality operator is ==, not =.

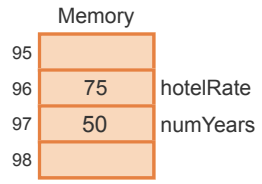
*Good practice is to indent a branch's statements, using a consistent number of spaces.* This material indents 3 spaces.

**PARTICIPATION ACTIVITY** 3.2.1: Detecting if two items are equal: Hotel discount.

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

int main() {
   int hotelRate;
   int numYears;

   hotelRate = 150;

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

   if (numYears == 50) {
      cout << "Congratulations on 50 years "
           << "of marriage!" << endl;

      hotelRate = hotelRate / 2;
   }

   cout << "Your hotel rate: ";
   cout << hotelRate << endl;
}
```

```
    cout << "Your hotel rate: ";
    cout << hotelRate << endl;

    return 0;
}
```

Enter numbers of years married: 50
Congratulations on 50 years of marriage!
Your hotel rate: 75






## Animation content:

Static figure:
Begin c++ code:

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

int main() {
  int hotelRate;
  int numYears;

  hotelRate = 150;

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

  if (numYears == 50) {
    cout << "Congratulations on 50 years "
         << "of marriage!" << endl;

    hotelRate = hotelRate / 2;
  }

  cout << "Your hotel rate: ";
  cout << hotelRate << endl;

  return 0;
}
```

End c++ code.
"Enter numbers of years married: 50
Congratulations on 50 years of marriage!
Your hotel rate: 75" is printed to the screen.

Step 1: The line of code reading "hotelRate = 150;" is highlighted, and hotelRate is assigned to 150.
The line of code reading "cout << "Enter number of years married: ";: is highlighted, and "Enter number of years married" is printed on the screen. The line of code reading " cin >> numYears;" is highlighted, the user enters 50, and numYears is assigned to 50.
Step 2: The if statement reading "if (numYears == 50)" is represented by "50 == 50" and is evaluated to be true.
Step 3: The code reading "cout << "Congratulations on 50 years " << "of marriage!" << endl;" is highlighted and "Congratulations on 50 years of marriage!" is printed to the screen.
"hotelRate = hotelRate / 2;" is highlighted, represented by "150/2", and hotelRate is assigned to 75.
Step 4: The ending closing bracket "}" is highlighted, and the if-statement exits.

Step 5: The lines of code reading "cout << "Your hotel rate: "; cout << hotelRate << endl;" are highlighted, and "Your hotel rate: 75" is printed to the screen.

### Animation captions:

1. An if statement executes a group of statements if an expression is true. The program assigns hotelRate with 150 and then gets the number of years the user has been married from input.
2. numYears is 50. So the expression numYears == 50 evaluates to true, and the if's statement will execute. Thus, the statement following the opening brace { will execute next.
3. hotelRate is divided in half, which is the discount for guests celebrating their 50th wedding anniversary.
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 = 10;
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

## Equality and inequality operators

Whereas the equality operator checks whether two values are equal, the ***inequality operator*** (***!=***) evaluates to true if the left and right sides are not equal, or different.

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



Table 3.2.1: Equality and inequality operators.

| Operator | Description | Example (assume x is 3) |
|---|---|---|

| | | |
|---|---|---|
| == | a **==** b means a is equal to b | x == 3 is true<br>x == 4 is false |
| != | a **!=** b means a is not equal to b | x != 3 is false<br>x != 4 is true |

**PARTICIPATION ACTIVITY** 3.2.3: 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** 3.2.4: 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

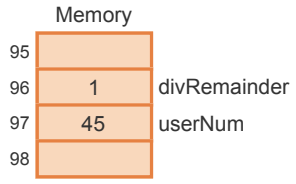


## If-else statement

An ***if-else*** statement executes one group of statements when an expression is true, and another group of statements when the expression is false. In the example below, the if-else statement outputs if a number entered by the user is even or odd. The if statement executes if divRemainder is equal to 0, and the else statement executes if divRemainder is not equal to 0.

**PARTICIPATION ACTIVITY** 3.2.5: If-else statement: Determining if a number is even or odd.

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

int main() {
   int userNum;
   int divRemainder;

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

   divRemainder = userNum % 2;

   if (divRemainder == 0) {
       cout << userNum << " is even." << endl;
   }
   else {
       cout << userNum << " is odd." << endl;
   }

   return 0;
}
```




```
Enter a number: 22
22 is even.

Enter a number: 45
45 is odd.
```

**Animation content:**

Static figure:
Begin c++ code:

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

int main() {
  int userNum;
  int divRemainder;

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

  divRemainder = userNum % 2;

  if (divRemainder == 0) {
     cout >> userNum >> " is even." >> endl;
  }
  else {
     cout >> userNum >> " is odd." >> endl;
  }

  return 0;
}
```

End c++ code.
"Enter a number: 22
22 is even.
Enter a number: 45
45 is odd." is shown on the screen.

Step 1: "cout >> "Enter a number: ";" is highlighted, and "Enter a number:" is printed. "Cin << userNum;" is highlighted, the user inputs the number 22, and userNum is assigned to 22.
Step 2: "divRemainder = userNum % 2;" is highlighted and represented by "22 % 2 = 0". divRemainder is set to 0.
Step 3: "if (divRemainder == 0)" is highlighted and evaluated to true. "cout >> userNum >> " is even." >> endl;" is highlighted, and "22 is even." is printed to the screen.
Step 4: "cout >> "Enter a number: ";" is highlighted, and "Enter a number:" is printed. "Cin << userNum;" is highlighted, the user inputs the number 45, and userNum is assigned to 45.
"divRemainder = userNum % 2;" is highlighted and represented by "45 % 2 = 1". divRemainder is set to 1.
"if (divRemainder == 0)" is highlighted and evaluated to false.
"cout >> userNum >> " is odd." >> endl" is highlighted, and "45 is odd." is printed to the screen.



## 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. userNum % 2 evaluates to the remainder of dividing userNum by 2. userNum is 22, so divRemainder is assigned with 0.
3. The if statement's expression divRemainder == 0 evaluates to 0 == 0, which is true. So the if's statements execute.

4. userNum is 45, so divRemainder is assigned with 1. The if statement's expression divRemainder == 0 evaluates to 1 == 0, which is false. So the else's statements execute.

**PARTICIPATION ACTIVITY** 3.2.6: If-else statements.

1) What is the final value of numItems?

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

Check Show answer



2) What is the final value of numItems?

```
bonusVal = 11;
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 == 14) {
   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 + 1;
}
else {
   bonusVal = bonusVal + 10;
}
```

Check Show answer

5) What is the final value of bonusVal?

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

Check Show answer



**PARTICIPATION ACTIVITY** 3.2.7: 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 equals 62, assign itemDiscount with 15. Else, assign itemDiscount with 0.

Check Show answer

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

Check Show answer

3) If numPlayers does not equal 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 branches with equality operators.

539740.3879454.qx3zqy7

Start

Type the program's output

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

int main() {
   int numPuppies;

   numPuppies = 5;

   if (numPuppies == 4) {
      cout << "a" << endl;
   }
   else {
      cout << "e" << endl;
   }

   cout << "k" << endl;

   return 0;
}
```

e
k

1 2

Check Next

**CHALLENGE ACTIVITY** 3.2.2: Basic if-else.



539740.3879454.qx3zqy7

Start

Write an if-else statement for the following:

If userTickets is not equal to 8, execute awardPoints = 10. Else, execute awardPoints = userTickets.

Ex: If userTickets is 14, then awardPoints = 10.

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

int main() {
   int awardPoints;
   int userTickets;

   cin >> userTickets; // Program will be tested with values: 6, 7, 8, 9

   /* Your code goes here */

   cout << awardPoints << endl;

   return 0;
}
```



1 2

Check Next

## Multi-branch if-else statements

Commonly, a program may need to detect several specific values of a variable. 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 statements execute (and no subsequent branch is considered). If no expression is true, the else branch ex

Figure 3.2.1: 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)
}
```



Figure 3.2.2: 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** 3.2.8: 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

## 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** 3.2.9: 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

**CHALLENGE ACTIVITY** 3.2.3: Detect specific values.



539740.3879454.qx3zqy7

Start

If integer numberOfYears is 1000, output "Equal to 1 millennium". Otherwise, output "Not equal to 1 millennium". End with a newline.

▶ **Click here for examples**

```
1 #include <iostream>
```

```
using namespace std;

int main() {
   int numberOfYears;

   cin >> numberOfYears;

   /* Your code goes here */

   return 0;
}
```



1 2 3

Check Next level

# 3.3 Detecting ranges with branches (general)

### Detecting ranges using if-elseif-else

A common programming task is to detect if a value lies within a certain range and then perform an action depending on where the value lies. Ex: If Timmy is less than 6, he can play pee-wee soccer. If Timmy is between 6 and 17, he can play junior league soccer, and if he's older than 17, he can play professional soccer.

An if-elseif-else structure can detect number ranges 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** 3.3.1: An if-elseif-else structure can elegantly detect ranges.

| | | | |
|---|---|---|---|
| 5 or under | No teams | If age < 6: | No teams |
| Under 8 | 6, 7 | Else If age < 8: | Play on U8 team |
| Under 10 | 8, 9 | Else If age < 10: | Play on U10 team |
| Under 12 | 10, 11 | Else If age < 12: | Play on U12 team |
| 12 or over | No teams | Else: | No teams |

### Animation content:

The following program is shown, conditions on the left, and pseudocode on the right.

Conditions:
5 or under: No teams.
Under 8: 6, 7.
Under 10: 8, 9.
Under 12: 10, 11.
12 or over: No teams.

Pseudocode:
If Age < 6: No teams.
Else If Age < 8: Play on U8 team
Else If age < 10: Play on U10 team
Else If age < 12: Play on U12 team
Else: No teams



### 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** 3.3.2: Using if-elseif-else to detect increasing ranges.

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

If unable to drag and drop, refresh the page.

**0 - 9** **20 - 29** **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** 3.3.3: More ranges with if-elseif-else.

Indicate the range detected by the expression, assuming each question continues a single if-elseif-else structure. 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



**CHALLENGE ACTIVITY** 3.3.1: Decision sequence to detect increasing ranges.

539740.3879454.qx3zqy7

Start

Indicate the smallest and largest numbers in the range detected by the first branch. Assume x is a non-negative integer.

If x < 13

// Range detected: Ex: 5 to Ex: 5

Else If x < 27

Else

1 2 3

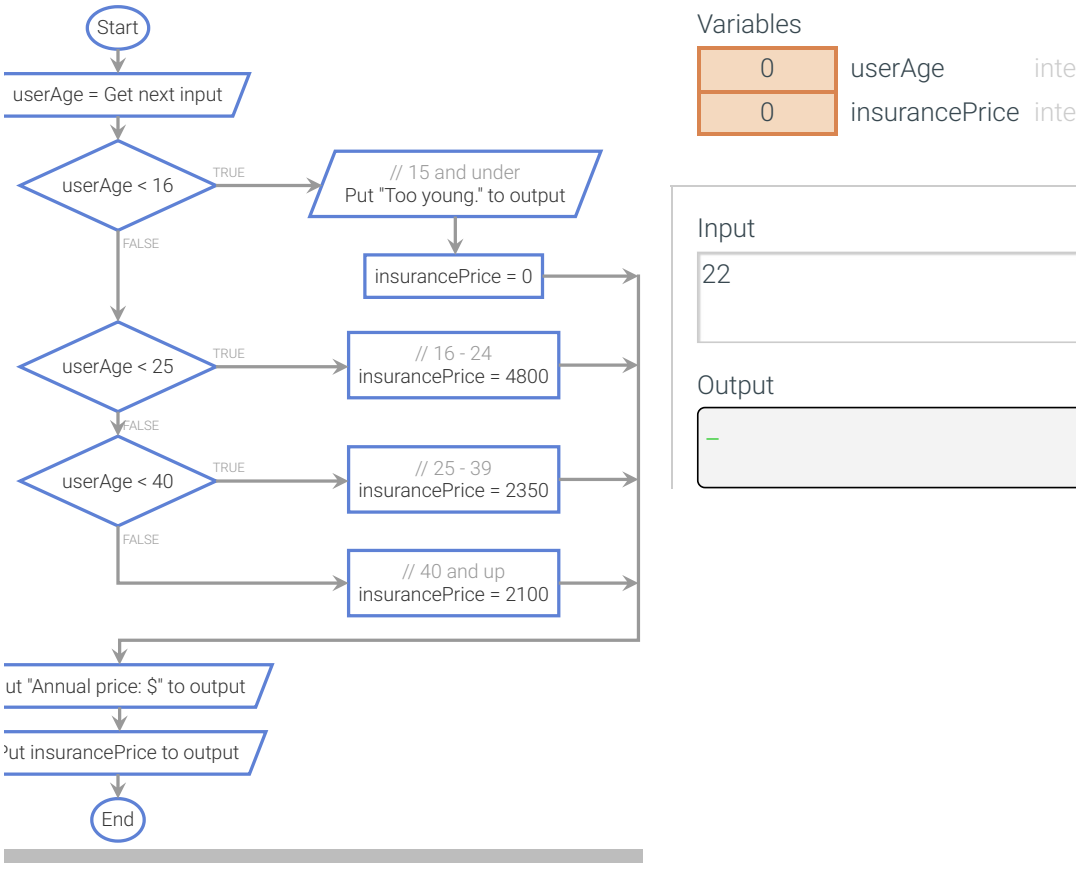
Check Next

## Using multi-branch if-else to detect ranges

The sequential nature of multi-branch if-else statements is useful 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).



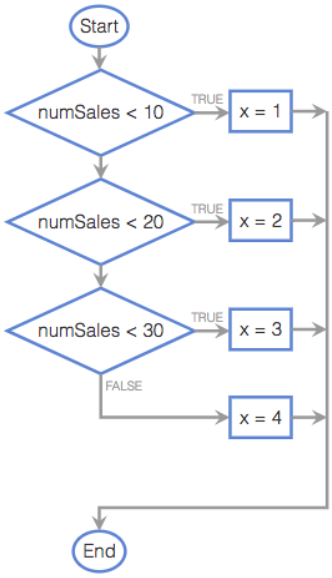
**PARTICIPATION ACTIVITY** 3.3.4: Using if-elseif for ranges: Insurance prices. Full screen




**PARTICIPATION ACTIVITY** 3.3.5: Decision sequences and ranges.



Type the range for each branch. Type ranges as 25 - 29, or as 30+ for 30 and up.






1) Range for x = 2

Check Show answer

2) Range for x = 3

Check Show answer

3) Range for x = 4

Check Show answer

**CHALLENGE ACTIVITY** 3.3.2: Flowchart decision sequence to detect increasing ranges.

539740.3879454.qx3zqy7

Start

Type the range for the given branch. Assume x is a non-negative integer.

Range for y = 1: Ex: 5 .. Ex: 5

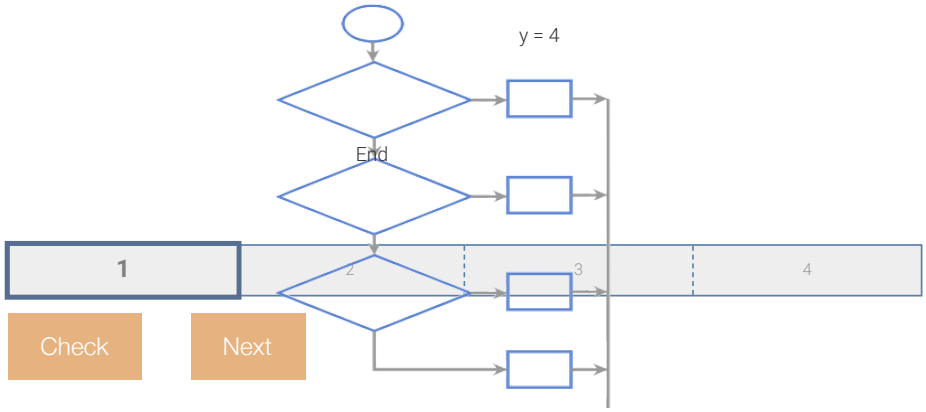


Start

x < 18 y = 1

x < 32 y = 2

x < 43 y = 3






# 3.4 Detecting ranges with branches

## 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 3.4.1: Relational operators.

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



**PARTICIPATION ACTIVITY** 3.4.1: 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** 3.4.2: 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



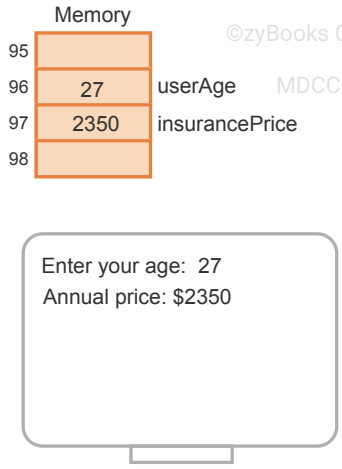
## Detecting ranges with if-else statements

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).

**PARTICIPATION ACTIVITY** | 3.4.3: Using the 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;
}
```






### Animation content:

Static figure:
Begin c++ code:
#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;
}
```

End c++ code.
"Enter your age: 27
Annual price: $2350" is printed on the screen.

Step 1: "cout << "Enter your age: "; cin >> userAge;" is highlighted, and the user enters 27. userAge is assigned to 27.
"if (userAge < 16)" is highlighted, and "27 < 16" evaluates to false.
Step 2: "else if (userAge < 25)" is highlighted, and "27 < 25" evaluates to false.

Step 3: "else if (userAge < 40)" is highlighted, and "27 < 40" evaluates to true. "insurancePrice = 2350;" is highlighted, and insurancePrice is set to 2350.
"cout << "Annual price: $" << insurancePrice << endl;" is highlighted, and "annual price: $2350" is printed.



### Animation captions:

1. The user enters 27 for their age, which is stored in memory as the variable userAge. The multi-branch if-else first checks if userAge is less than 16, which is false.
2. The next if statement in the multi-branch if-else checks if userAge is less than 25, which is false.
3. The next branch checks if userAge is less than 40, which is true. The else if's statements execute and the variable insurancePrice is set to 2350 in memory.

**PARTICIPATION ACTIVITY** 3.4.4: 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.

```
int numSales;

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?

```
int numItems;

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

Check Show answer

**PARTICIPATION ACTIVITY** 3.4.5: Complete the multi-branch if-else.

1) Second branch: userNum is less than 200

```
int userNum;

if (userNum < 100 ) {

   ...

}

else if (          )
{

   ...

}

else { // userNum >= 200

   ...

}
```

Check Show answer

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

```
int userNum;
if (userNum < 0 ) {
   ...
}
[ ] {
   ...
}
else { // userNum is 0
   ...
}
```

Check Show answer

3) Second branch: userNum is greater than 105

```
int userNum;
if (userNum < 100 ) {
   ...
}
[ ] {
   ...
}
else { // userNum is between
      // 100 and 105
   ...
}
```

Check Show answer



©zyBooks 01/31/24 17:45 1939727
Rob Daglio
MDCCOP2335Spring2024

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

```
int userNum;

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.

```
int userNum = 555;

if (userNum < 0) {
   ... // Branch 1
}
else if (userNum > 600) {
   ... // Branch 2
}
else if (userNum < 100) {
   ... // Branch 3
}
```

Check Show answer



**CHALLENGE ACTIVITY** 3.4.1: Detect ranges using branches.

539740.3879454.qx3zqy7

Start

Type the program's output

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

int main() {
   int x;

   cin >> x;

   if (x > 15) {
      cout << "e" << endl;
   }
   else {
      cout << "w" << endl;
   }

   cout << "p" << endl;

   return 0;
}
```

Input

14

Output

```
w
p
```



1 2 3 4 5

Check Next

**CHALLENGE ACTIVITY** 3.4.2: Basic if-else expressions.

539740.3879454.qx3zqy7

**Start**

Complete the if-else statement to output "Greater than 50" if the value of userNum is greater than 50. Otherwise, output "50 or less".

Ex: If the input is `52`, then the output is:

```
Greater than 50
```

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     int userNum;
6
7     cin >> userNum;
8
9     if (/* Your code goes here */) {
10       cout << "Greater than 50" << endl;
11    }
12    else {
13       cout << "50 or less" << endl;
14    }
15
```



1 2

Check Next level

**CHALLENGE ACTIVITY** 3.4.3: Working with branches.

539740.3879454.qx3zqy7

Start

Integer carCount is read from input. If carCount is more than 16, then output "Way too many cars". End the output with a nev

▶ **Click here for examples**

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

int main() {
   int carCount;

   cin >> carCount;

   /* Your code goes here */

   return 0;
}
```



1 2 3 4

Check Next level

**CHALLENGE ACTIVITY** 3.4.4: Working with ranges.

539740.3879454.qx3zqy7

Start

When the input integer variable outfitCount is:

- less than 7, output "Small suitcase".
- between 7 inclusive and 18 exclusive, output "Large suitcase".
- greater than or equal to 18, output "Need more than one suitcase".

End with a newline.

▶ **Click here for examples**

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

int main() {
   int outfitCount;

   cin >> outfitCount;

   /* Your code goes here */

   return 0;
}
```

1 2

Check Next level



# 3.5 Detecting ranges using 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** 3.5.1: Logical operators: AND, OR, and NOT.

| a | b | a AND b |
|---|---|---|
| false | false | false |
| false | true | false |
| true | false | false |
| true | true | true |

| a | b | a OR b |
|---|---|---|
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | true |

| a | NOT a |
|---|---|
| false | true |
| true | false |

*Let x = 7, y = 9*

(x > 0) AND (y < 10) → true (true AND true)

(x > 0) AND (y < 5) → false (true AND false)

(x < 0) OR (y > 10) → false (false OR false)

(x < 0) OR (y > 5) → true (false OR true)

NOT (x < 0) → true (NOT false)

NOT (x > 0) → false (NOT true)

### Animation content:

Three truth tables are shown, each evaluating a different expression.

The first truth table takes in operands a and b and then uses the AND operator. False AND false evaluates to false. False AND true evaluates to false. True AND false evaluates to false. True AND true evaluates to true.



An example below is shown where x=7 and y=9. The expression reads (x > 0) AND (y < 10). Both operands are true so the entire expression evaluates to true. The next expression is (x > 0) AND (y < 5). The first operand is true, but the second is false, so the entire expression evaluates to false.

The second truth table takes in operands a and b and then uses the OR operator. False OR false evaluates to false. False OR true evaluates to true. True OR false evaluates to true. True OR true evaluates to true.

An example below is shown where x=7 and y=9. The expression reads (x < 0) OR (y > 10). Both operands are false so the entire expression evaluates to false. The next expression is (x < 0) AND (y > 5). The first operand is false, but the second is true, so the entire expression evaluates to true.

The third truth table takes in operand a and then uses the NOT operator. False evaluates to true. True evaluates to false.

An example below is shown where x=7 and y=9. The expression reads NOT(x < 0). The operand alone evaluates to true, but the expression as a whole evaluates to false. The expression reads NOT(x > 0). The operand alone evaluates to false, but the expression as a whole evaluates to true.



### 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 3.5.1: 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** 3.5.2: 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

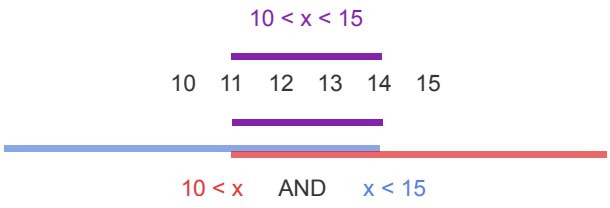
7) NOT ( (x > 5) AND (y < 20) )

- 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** 3.5.3: Using AND to detect if a value is within a range.




### Animation content:

At the top, a purple line represents the wanted range 10 < x < 15. Underneath this purple line are the numbers 10, 11, 12, 13, 14, and 15. The purple line only exists between 11 and 14.



Further below there is a red line that starts at 11 and stretches to the right to represent the range (10 < x): all values greater than 10.
Above the red line, there is a blue line that starts at 14 and stretches to the left to represent the range (x < 15): all values less than 15.

The AND operator appears and combines the two range expressions (10 < x AND x < 15) and a second purple line is formed from where the red and blue lines overlap. The second purple line only

exists between 11 and 14 to represent the range (10 < x) and (x < 15). Note: 10 < x < 15 is not a valid expression in Java. 10 < x AND x < 15 is a correct way to identify the 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** 3.5.4: 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. Logical operators are commonly used in expressions of if-else statements.

Table 3.5.2: 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** 3.5.5: Logical operators.

Match the symbol with the logical operator.

If unable to drag and drop, refresh the page.

! || !! &&



| | |
|---|---|
| | AND |
| | OR |
| | NOT |
| | No such operator |

Reset

**PARTICIPATION ACTIVITY** 3.5.6: 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** 3.5.7: 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

*A common error is to use & instead of && and | instead ||. & and | are not logical operators and may produce unexpected output.*

**PARTICIPATION ACTIVITY** 3.5.8: Logical expression simulator.



Try typing different expressions involving x, y and observe whether the expression evaluates to true for different values of x, y.
Ex: Test the expression (x < 10) && (x > 2) when x is assigned with 1, 2, 5, and 20.

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

Run code

**Output is:**

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

## 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 3.5.1: Detecting ranges: Cable TV channels.

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



zyDE 3.5.1: 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
11       channelType = 's';
12    }
13    else if ((userChannel
14       channelType = 'h';
15    }
```

3

Run



**PARTICIPATION ACTIVITY** 3.5.9: 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 paid channels. Does this expression detect that range? `(userChannel >= 500) || (userChannel <= 599)`
   - Yes
   - No



## Detecting ranges implicitly vs. explicitly

A programmer often uses logical operators to detect a range by explicitly specifying the high-end and low-end of the range. However, 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.

**PARTICIPATION ACTIVITY** 3.5.10: Detecting ranges implicitly vs. explicitly.



```
if (x < 0) {
 // Negative
}
else if ( (x >= 0) && (x <= 10) ) {
 // 0..10
}
else if ( (x >= 11) && (x <= 20) ) {
 // 11..20
}
else {
 // 21+
}
```

*Explictly defined ranges*

```
if (x < 0) {
 // Negative
}
else if (x <= 10) {    x >= 0 is implicit
 // 0..10
}
else if (x <= 20) {    x > 10 is implicit
 // 11..20
}
else {                 x > 20 is implicit
 // 21+
}
```

*Implictly defined ranges*

### Animation content:

On the left side is an example of explicitly defined ranges. The example reads as follows, with comments defining the ranges:

```
if (x < 0) {
 // Negative
}
else if ( (x >= 0) && (x <= 10) ) {
 // 0..10
}
else if ( (x >= 11) && (x <= 20) ) {
 // 11..20
}
else {
 // 21+
}
```

On the right side is an example of implicitly defined ranges. The example reads as follows, with comments defining the ranges:



```
if (x < 0) {
 // Negative
}
else if (x <= 10) {
 // 0..10
}
else if (x <= 20) {
 // 11..20
}
else {
```

```
 // 21+
}
```

In this example x < 0, x <= 10, and x <= 20 are all implicitly defining their respective ranges. Both the explicit range example on the left and the implicit range example on the right define the same range.

### 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. Implicit ranges can simplify a multi-branch if statement for ranges without gaps.



**PARTICIPATION ACTIVITY** 3.5.11: Detecting ranges implicitly vs explicitly.

For each pair of statements, does the second if-else statement detect the same ranges as the first if-else statement?

1)

```
if (temp <= 0)...
else if ((temp > 0) && (temp <
100))...
```

```
if (temp <= 0)...
else if (temp < 100)...
```

- Yes
- No

2)

```
if (systolic < 130)...
else if ((systolic >= 130) &&
(systolic <= 139))...
```

```
if (systolic < 130)...
else if (systolic >= 130)...
```

- Yes
- No

3)

```
if ( (year >= 1901) && (year <=
2000) )...
else if ((year >= 2001) && (year
<= 2100))...
```

```
if (year <= 2000)...
else if (year <= 2100)...
```

- Yes
- No



**CHALLENGE ACTIVITY** 3.5.1: Detecting ranges using logical operators.

539740.3879454.qx3zqy7

Start

Modify the given if statement so that "Different tax bracket" is output if salaryInput is outside the range 54000 - 84000 inclus
Otherwise, "22% tax bracket" is output.

▶ **Click here for examples**

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

int main() {
   int salaryInput;

   cin >> salaryInput;

   // Modify the following line
   if ((salaryInput < 54000) && (salaryInput > 84000)) {
      cout << "Different tax bracket" << endl;
   }
   else {
      cout << "22% tax bracket" << endl;
   }
```



1 2 3 4

Check Next level

# 3.6 Detecting ranges with gaps

## Basic ranges with gaps

Oftentimes, ranges contain gaps. Ex: Movie theaters often give ticket discounts to children (anyone 12 and under) and seniors (anyone 65 and older). The gap is the group of people aged 13 to 64. An if-else statement can be used to detect such ranges with gaps.

**PARTICIPATION ACTIVITY** | 3.6.1: Using multi-branch if-else for detecting ranges with gaps: Movie ticket prices.



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

int main() {
   int userAge;
   int movieTicketPrice;

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

   if (userAge <= 12) {            // Age 12 and under
      cout << "Child ticket discount." << endl;
      movieTicketPrice = 11;
   }
```

Memory

| | | |
|---|---|---|
| 95 | | |
| 96 | 19 | userAge |
| 97 | 14 | movieTicketPrice |
| 98 | | |

```
Enter your age:  67
Senior ticket discount.
```

```
   else if (userAge >= 65) {       // Age 65 and older
      cout << "Senior ticket discount." << endl;
      movieTicketPrice = 12;
   }
   else {                          // All other ages
      movieTicketPrice = 14;
   }

   cout << "Movie ticket price: $"
        << movieTicketPrice << endl;
   return 0;
}
```

```
Movie ticket price: $12

Enter your age: 19
Movie ticket price: $14
```

67 <= 12 X
67 >= 65 ✓

19 <= 12 X
19 >= 65 X



### Animation content:

Static image:
There are one block of code, an associated section of memory showing variable values after the last execution, a monitor that displays inputs and outputs, and the results of four numerical comparisons.

Begin C++ code:

```
using namespace std;

int main() {
  int userAge;
  int movieTicketPrice;

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

  if (userAge <= 12) {          // Age 12 and under
    cout << "Child ticket discount." << endl;
    movieTicketPrice = 11;
  }
  else if (userAge >= 65) {     // Age 65 and older
    cout << "Senior ticket discount." << endl;
    movieTicketPrice = 12;
  }
  else {                        // All other ages
    movieTicketPrice = 14;
  }

  cout << "Movie ticket price: $"
       << movieTicketPrice << endl;
  return 0;
```

End C++ code.



The display has five lines of input/output: "Enter your age: 67", "Senior ticket discount.", and "Movie ticket price: $12", after which is an extra line break, then the lines "Enter your age: 19" and "Movie ticket price: $14".

The memory shows that "19" is stored in the userAge variable and "14" is stored in the movieTicketPrice variable.

These are the results of comparisons that also are visible, with "X" signifying false and "Y" signifying

true: "67 <= 12 X", "67 >= 65 Y", "19 <= 12 X", and "19 >= 65 X".

Step 1: After the user enters their age, the else-if branch's first branch checks if age is <= 12. Memory is empty and the display is blank. In the code, 'cout << "Enter your age: ";' is highlighted and "Enter your age: " is printed to the display. In the code, "cin >> userAge;" is highlighted. After the user inputs "67", the value"67" is stored at the address of userAge in memory. In the code, "if (userAge <= 12) { // Age 12 and under" is highlighted.

Step 2: userAge is 67, which is greater than 12, so the program moves to the second branch that checks if userAge is >= 65. In the code, "else if (userAge >= 65) { // Age 65 and older" is highlighted.

Step 3: 67 is >= 65, so the second branch's statements execute, applying the senior discount to the ticket price. The program concludes by outputting the ticket price. In the code, "cout << "Senior ticket discount." << endl;" is highlighted and "Senior ticket discount" is printed to the display. In the code, " movieTicketPrice = 12;" is highlighted and the value "12" is stored at the movieTicketPrice location in memory. In the code, 'cout << "Movie ticket price: $"
<< movieTicketPrice << endl;' is highlighted and "Movie ticket price: $12'" is printed to the display.

Step 4: If the user's age falls between the gap of 12 and 65 (13 to 64), the else branch executes and the ticket price is $14, the most expensive price. In the code, 'cout << "Enter your age: ";' is highlighted and "Enter your age: " is printed to the display. In the code, "cin >> userAge;" is highlighted. After the user inputs "19", the value "19" is stored at the address of userAge in memory. In the code, "if (userAge <= 12) { // Age 12 and under" is highlighted. Since 19 is not <= 12, the if statement's body is skipped. In the code, "else if (userAge >= 65) { // Age 65 and older" is highlighted. Since 19 is not >= 65, the if statement's body is skipped. In the code, "else { // All other ages" is highlighted and then "movieTicketPrice = 14;" is highlighted. The value "14" is stored at the address of movieTicketPrice in memory. In the code, 'cout << "Movie ticket price: $"
<< movieTicketPrice << endl;' is highlighted and "Movie ticket price: $14" is printed to the display. Execution ends.

### Animation captions:

1. After the user enters their age, the else-if branch's first branch checks if age is <= 12.
2. userAge is 67, which is greater than 12, so the program moves to the second branch that checks if userAge is >= 65.
3. 67 is >= 65, so the second branch's statements execute, applying the senior discount to the ticket price. The program concludes by outputting the ticket price.
4. If the user's age falls between the gap of 12 and 65 (13 to 64), the else branch executes and the ticket price is $14, the most expensive price.

**PARTICIPATION ACTIVITY** 3.6.2: Detecting ranges with gaps and multi-branch if-else.



Select the correct answers below.

1) In the animation above, what is the age range for a child ticket discount?
   - 0 - 12
   - less than 13
   - less than 11

2) In the animation above, what is the age range for a senior ticket discount?

- 65 or more
- 66 or more
- 13 - 64

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

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

- 1 - 99
- 0 - 100
- 1 - 100

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

```
if (numItems < 50) {
   ...
}
else if (numItems > 50) {
   ...
}
else {  // Range: _______
   ...
}
```

- 49 - 51
- 0 - 50
- 50



## Ranges with gaps using logical operators

Programmers often use logical operators to explicitly detect ranges with an upper and lower bound, including ranges with gaps that may have intermediate bounds. Ex: If a valid office number is within the ranges of 100 to 150 or 200 to 250, the logical AND operator can be used to identify the lower and upper bounds of the two ranges. Further, the ranges can be combined into a single branch using the logical OR operator.

**PARTICIPATION ACTIVITY** 3.6.3: Explicit ranges with gaps detection using logical AND and OR.



```
if (officeNum >= 100 && officeNum <= 150) {
   // valid office number
}
else if (officeNum >= 200 && officeNum <= 250) {
   // valid office number
}
else {
   // invalid office number
}
```

```
if ((officeNum >= 100 && officeNum <= 150) || (officeNum >= 200 && officeNum <= 250)) {
   // valid office number
}
else {
   // invalid office number
}
```

## Animation content:



Static image:
There are two blue boxes with code.

The first blue box:
Begin C++ code:

```
if (officeNum >= 100 && officeNum <= 150) {
  // valid office number
}
else if (officeNum >= 200 && officeNum <= 250) {
  // valid office number
}
else {
  // invalid office number
}
```

End C++ code.

The following lines of code from the first blue box are highlighted:

```
if (officeNum >= 100 && officeNum <= 150) {
  // valid office number
```

The following lines of code from the first blue box are also highlighted:

```
else if (officeNum  >= 200 && officeNum <= 250) {
  // valid office number
```

The second blue box :
Begin C++ code:

```
if ((officeNum >= 100 && officeNum <= 150) || (officeNum >= 200 && officeNum <= 250)) {
  // valid office number
}
else {
  // invalid office number
}
```

End C++ code.



The following lines of code from the second blue box are highlighted:

```
if ((officeNum >= 100 && officeNum <= 150) || (officeNum >= 200 && officeNum <= 250)) {
  // valid office number
```

Step 1: The logical AND operator is used to identify the lower and upper bounds of the two valid ranges of office numbers (100 to 150 and 200 to 250). Any number outside of the ranges is in the gap. The following lines of code from the first blue box are highlighted :

```
if (officeNum >= 100 && officeNum <= 150) {
```

```
  // valid office number
```

The following lines of code from the first blue box are also highlighted:

```
else if (officeNum  >= 200 && officeNum <= 250) {
  // valid office number
```

Step 2: Further, the two ranges can be combined into a single branch using the logical OR operator. The following lines of code from the second blue box are highlighted:

```
if ((officeNum >= 100 && officeNum <= 150) || (officeNum >= 200 && officeNum <= 250)) {
  // valid office number
```



### Animation captions:

1. The logical AND operator is used to identify the lower and upper bounds of the two valid ranges of office numbers (100 to 150 and 200 to 250). Any number outside of the ranges is in the gap.
2. Further, the two ranges can be combined into a single branch using the logical OR operator.

**PARTICIPATION ACTIVITY** 3.6.4: NFL Jersey numbers.

In the National Football League (NFL), player positions have jersey numbers in specific ranges.
Ex: An NFL wide receiver can only wear jersey numbers from 10 to 19 or 80 to 89.
Select the if statement that explicitly detects the correct NFL jersey number ranges.

1) Linebacker: 40 to 59 or 90 to 99

```
if ((jNum >= 40 && jNum <= 59) || (jNum >= 90 && jNum <= 99))
```

```
if ((jNum > 40 && jNum <= 59) || (jNum > 90 && jNum <= 99))
```

```
if (jNum >= 40 && jNum <= 99)
```

2) Tight end: 40 to 49 or 80 to 89

```
if ((jNum >= 40 && jNum <= 49) && (jNum >= 80 && jNum <= 89))
```

```
if ((jNum >= 40 || jNum <= 49) && (jNum >= 80 || jNum <= 89))
```

```
if ((jNum >= 40 && jNum <= 49) || (jNum >= 80 && jNum <= 89))
```

3) Defensive lineman: 50 to 79 or 90 to 99



```
if ((jNum > 50 && jNum < 79) || (jNum > 90 && jNum < 99))
```

```
if ((jNum >= 49 && jNum =< 80) || (jNum >= 89 && jNum =< 100))
```

```
if ((jNum > 49 && jNum < 80) || (jNum > 89 && jNum < 100))
```

4) Quarterback: 1 to 19

- `if (jNum <= 19)`
- `if (jNum > 0 && jNum < 20)`
- `if (jNum > 0 || jNum < 20)`

**CHALLENGE ACTIVITY** 3.6.1: Enter the output of the branch expressions.



539740.3879454.qx3zqy7

Start

Type the program's output

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

int main() {
   int x;

   x = 3;
   if ( (x < 6) && (x > 1) ) {
      cout << "a" << endl;
   }
   else {
      cout << "b" << endl;
   }

   return 0;
}
```

a

1 2 3 4 5

Check Next

**CHALLENGE ACTIVITY** 3.6.2: Ranges with gaps.

539740.3879454.qx3zqy7

Start

Integer currentPlums is read from input representing the number of plums. If the number of plums is fewer than or equal to 30 or more, output "Unsatisfactory quantity", followed by a newline.

▶ **Click here for example**

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

1 2 3 4

Check Next level



# 3.7 Detecting multiple features with branches

### Multiple distinct if statements

A programmer can use multiple if statements in sequence to detect multiple features with independent actions. Multiple sequential if statements 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 3.7.1: 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** 3.7.1: 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** 3.7.1: Enter the output for the multiple if-else branches.

539740.3879454.qx3zqy7

Start

Type the program's output

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

int main() {
   int numItems;

   numItems = 5;

   if (numItems < 3) {
      cout << "a" << endl;
   }
   else if (numItems < 9) {
      cout << "e" << endl;
   }
   else {
      cout << "h" << endl;
   }

   cout << "r" << endl;

   return 0;
}
```

er

1 2 3 4

Check Next



## 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. Nested if statements are commonly used to make decisions that are based on multiple features. Ex: To calculate a discount based on both the number of items purchased and the total cost of those items, one if statement checks the number of items purchased and a nested if statement can check the total cost.

Figure 3.7.2: Nested if-else.

```
if (numItems > 3) {
   if (totalCost > 100) {       // numItems > 3 and totalCost > 100
      saleDiscount = 20;
   }
   else if (totalCost > 50) {  // numItems > 3 and totalCost > 50
      saleDiscount = 10;
   }
}
else if (numItems > 0) {
   ...
}
```



**PARTICIPATION ACTIVITY** 3.7.2: 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



**CHALLENGE ACTIVITY** 3.7.2: Detecting multiple features with branches.

539740.3879454.qx3zqy7

Start

Integer numMinutes is read from input. Write multiple if statements:

- If numMinutes is greater than 11, then output "I can finish a third of my homework."
- If numMinutes is greater than 53, then output "I can finish my homework and have extra time."
- If numMinutes is less than 7, then output "I don't have time to do my homework."

End each output with a newline.

▶ **Click here for examples**



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

int main() {
   int numMinutes;

   cin >> numMinutes;

   /* Your code goes here */

   return 0;
}
```

1 2 3

Check Next level

# 3.8 Common branching errors

## Common error: Missing braces

*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** 3.8.1: Common error when omitting braces.



```
if (numSales < 20)    15 < 20
   salesBonus = 0;
else
   totBonus = totBonus + 1;
   salesBonus = 20;
```

Memory

| | | |
|---|---|---|
| 95 | | |
| 96 | 15 | numSales |
| 97 | 20 | salesBonus |
| 98 | 2 | totBonus |

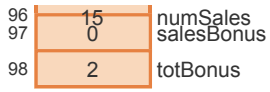
*Indentation is irrelevant.*
*salesBonus = 20; is not part of else,*
*so always executes.*

```
if (numSales < 20) { 15 < 20
   salesBonus = 0;
```

Memory

| | |
|---|---|
| 95 | |

```
}
else {
   totBonus = totBonus + 1;
   salesBonus = 20;
}
```




*Always using braces avoids the above common error.*

### Animation content:



Static figure:

Begin C++ code:
if (numSales < 20)
  salesBonus = 0;
else
  totBonus = totBonus + 1;
  salesBonus = 20;
End C++ code.
Atop the code is a note that numSales is less than 20.
  Underneath the code is some text. It reads: Indentation is irrelevant. salesBonus = 20; is not part of else, so the assignment statement always executes.

Memory is to the right, with 3 locations filled by the 3 variables presented in the code. In slot 96 is 15, the numSales variable. In slot 97 is 20, the salesBonus variable; this value of 20 is highlighted to indicate incorrectness. In slot 98 is 2, the totBonus variable.

Below this example is the corrected code snippet using braces:

Begin C++ code:
if (numSales < 20) {
  salesBonus = 0;
}
else {
  totBonus = totBonus + 1;
  salesBonus = 20;
}
End C++ code.

Atop the code is a note that numSales is less than 20.
 Underneath the code is some text. It reads: Always using braces avoids the above common error.

To the right is memory with the correct values. In slot 96 is 15, the numSales variable. In slot 97 is 0, the salesBonus variable. In slot 98 is 2, the totBonus variable.



### 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** 3.8.2: 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** 3.8.1: 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;
```



Learn how our autograder works

539740.3879454.qx3zqy7

```
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



## Common error: Using the incorrect operators

*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** 3.8.3: 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

**CHALLENGE ACTIVITY** 3.8.2: If-else statement: Fix errors.

539740.3879454.qx3zqy7



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
```

```
   cin >> userNum;    // Program will be tested with values: 1, 2, 3, 0.

   if (userNum = 2) {
      cout << "Num is greater or equal to two" << endl;
   }
   else {
      cout << "Num is less than two" << endl;
   }

   return 0;
}
```



1

Check Try again

# 3.9 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 people 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 3.9.1: 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 3.9.1: 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** 3.9.1: 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 3.9.2: 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 3.9.2: 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 6:59pm
         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
```

```
   // 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** 3.9.2: 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 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 people in the vehicle. The carpool discount rules are:

- A carpool is 3 or more people 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 3.9.3: 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 numPeople;    // People 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 people in vehicle: ";
   cin >> numPeople;

   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 people) and update toll
   if (numPeople >= 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 people in vehicle: 3
Toll at 5:15 pm is 0
```



**PARTICIPATION ACTIVITY** 3.9.3: Toll calculation.

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

If unable to drag and drop, refresh the page.

4.65 | 1.55 | 0.0 | 2.325

| | |
|---|---|
| | 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

# 3.10 Order of evaluation

## Precedence rules

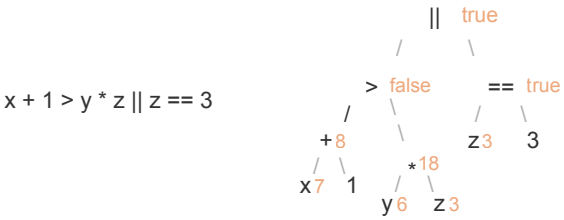


The order in which operators are evaluated in an expression are known as ***precedence rules***. Arithmetic, logical, and relational operators are evaluated in the order shown below.

Table 3.10.1: 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 &&.<br>== 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** | 3.10.1: Applying the precedence rules to an expression can be thought of as a 'tree'.






## Animation content:

Static Figure:
To the left is an expression that reads: x + 1 > y * z || z == 3

To the right is a graphical representation of expression as a tree, with text explaining what each node ultimately evaluates to.
The root is "||" and evaluates to true.
The left child of "||" is ">" and evaluates to false.
The right child of "||" is "==" and evaluates to true.
The left child of ">" is "+" and evaluates to 8.
The right child of ">" is "*" and evaluates to 18.
The left child of "==" is "Z", is a leaf node, and evaluates to 3
The right child of "==" is "3" and is a leaf node.
The left child of "+" is "x", is a leaf node, and evaluates to 7.
The right child of "+" is "1" and is a leaf node.
The left child of "*" is "y", is a leaf node, and evaluates to 6.
The right child of "8" is "z", is a leaf node, and evaluates to 3.

Step 1: Expressions like x + 1 > y * z || z == 3 are evaluated using precedence rules. Among +, >, *, ||, and ==, the * comes first.
"y * z" is highlighted.
Step 2: Next comes +, then >, then ==, and finally ||.
"x + 1" is highlighted, followed by "> y", then "z == 3", and finally "z || z".
Step 3: The expression is actually treated like a "tree", evaluated from the bottom upwards.
A graphical representation of the tree is shown.
The root is "||".
The left child of "||" is ">".
The right child of "||" is "==".
The left child of ">" is "+".
The right child of ">" is "*".
The left child of "==" is "Z" and is a leaf node.
The right child of "==" is "3" and is a leaf node.
The left child of "+" is "x" and is a leaf node.
The right child of "+" is "1" and is a leaf node.
The left child of "*" is "y" and is a leaf node.
The right child of "8" is "z" and is a leaf node.
First, "*" is highlighted, followed by "+", ">", "==", and "||".
Step 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.

First, y and z are represented by 6 and 3 in the tree. Their parent, "*", evaluates to 18. X is then represented as 7, and its parent, "+", is evaluated with the sibling, "1", to be 8.
The parent of these two operations, ">", evaluates to false.
z is represented by 3 again, and its parent, "==" is evaluated with the sibling, "3", to be true.
Finally, "false" and "true" are compared with their parent, "||" and evaluated to be true.

### 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** 3.10.2: 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** 3.10.3: 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** 3.10.4: 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** 3.10.5: 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** 3.10.6: 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



# 3.11 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** 3.11.1: Switch statement.

```
switch (a) {
  case 0:
     // Print "zero"
     break;

  case 1:
     // Print "one"
     break;

  case 2:
     // Print "two"
     break;

   default:
     // Print "unknown"
     break;
}
```

Output

a = 1 → one

a = 5 → unknown

## Animation content:

Static figure:
Begin c++ code:
switch (a) {
 case 0:
  // Print "zero"
  break;

 case 1:
  // Print "one"
  break;

 case 2:

```
    // Print "two"
    break;

  default:
    // Print "unknown"
    break;
}
```

End C++ code.

Two output boxes are shown. In the first, variable a equals 1. The screen displays "one." In the second, variable a equals 5. The screen displays "unknown."



Step 1: A switch statement can more clearly represent multi-branch behavior involving a variable being compared to constant values.
The lines of code reading "case 0:", "case 1:", "case 2:", and "default:" are highlighted.
Step 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.
Variable a is set to 1. The code reading "case 0:", and then "case 1:" are highlighted. "//Print "one" " is then highlighted, and "one" is printed to the screen. Next, "break;" is highlighted, and the program exits.
Step 3: If no case matches, then the default case statements are executed.
Variable a is set to 5. In order, the following lines of code are highlighted: "case 0:", "case 1:", "case 2:", "default:". Next, "//Print "unknown"" is highlighted, "unknown" is printed to the screen, "break;" is highlighted, and the program exits.

### 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** 3.11.2: 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 3.11.1: Switch statement general form.

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

   case constantExpr2:
      // Statements
      break;

   ...

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



Figure 3.11.1: Switch example: Estimates a dog's age in human years.

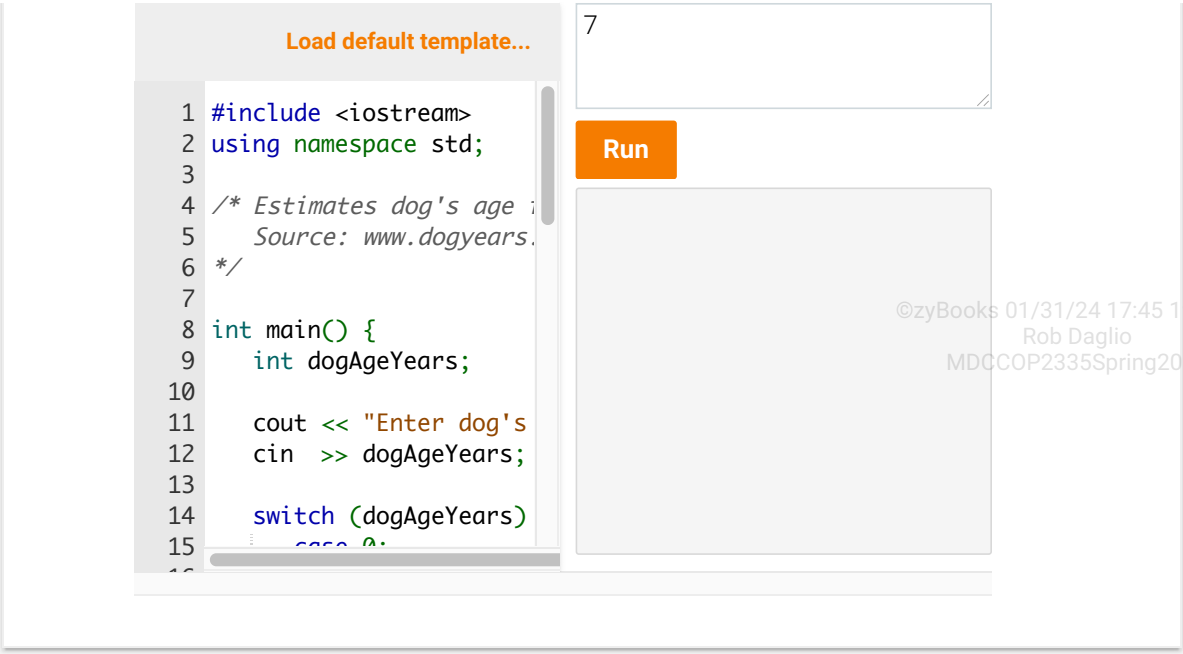
```
#include <iostream>
using namespace std;

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

int main() {
   int dogAgeYears;

   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.
```



## zyDE 3.11.1: Switch statement: Numbers to words.

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




## Omitting the break statement

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 3.11.2: Switch example: Dog years with months.

```
#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;
}
```

```
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.
```



©zyBooks 01/31/24 17:45 1939727
Rob Daglio
MDCCOP2335Spring2024

**PARTICIPATION ACTIVITY** 3.11.3: 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 | 3.11.1: Rock-paper-scissors. Full screen

539740.3879454.qx3zqy7

Organize the lines of a switch statement that checks nextChoice.

If the input is 0, the output is:

```
Rock
```

If the input is 1, the output is:

```
Paper
```

If the input is 2, the output is:

```
Scissors
```

For any other value, output "Unknown".



How to use this tool

**Unused**

```
case 0:
cout << "Rock" << endl;
break;
default:
cout << "Unknown" << endl;
}
cout << "Scissors" << endl;
break;
case 1:
switch (nextChoice){
cout << "Paper" << endl;
break;
case 2:
```

**main.cpp** Load default templ

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

int main() {
   int nextChoice;

   cin >> nextChoice;

   return 0;
}
```

Check

**CHALLENGE ACTIVITY** 3.11.2: 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.

Learn how our autograder works

539740.3879454.qx3zqy7

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

Run

# 3.12 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: `isWeekend = true;` assigns isWeekend with the Boolean value true. A Boolean variable may also be set to the result of a logical expression. Ex: `isLargeParty = (partySize > 6);` assigns isLargeParty with the result of the expression partySize > 6.

Figure 3.12.1: Variables of bool data type: Restaurant wait time calculator.

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

int main() {
   int waitTime;
   int partySize;
   char day;
   bool isLargeParty;
   bool isWeekend;

   // Get day of reservation
   cout << "Day of reservation (T/W/R/F/S/U): ";
   cin >> day;
   if (day == 'F' || day == 'S' || day == 'U') {
      isWeekend = true;
   }
   else {
      isWeekend = false;
   }

   // Get party size
   cout << "Enter party size: ";
   cin  >> partySize;
   isLargeParty = (partySize > 6);

   // Determine wait time based on day of week and party size
   if (isWeekend && !isLargeParty) {
      waitTime = 30;
   }
   else if (!isWeekend && !isLargeParty) {
      waitTime = 10;
   }
   else if (isWeekend && isLargeParty) {
      waitTime = 45;
   }
   else {
      waitTime = 15;
   }

   cout << "Restaurant wait time is " << waitTime
        << " minutes." << endl;

   return 0;
}
```

```
Day of reservation (T/W/R/F/S/U): W
Enter party size: 5
Restaurant wait time is 10 minutes.

...

Day of reservation (T/W/R/F/S/U): U
Enter party size: 10
Restaurant wait time is 45 minutes.
```



**PARTICIPATION ACTIVITY** 3.12.1: 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 3.12.2: 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** 3.12.2: 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 or equal to 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** 3.12.1: Using bool.

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

Learn how our autograder works

539740.3879454.qx3zqy7

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     bool isTeenager;
6     int kidAge;
7
8     cin >> kidAge;
9
10    /* Your solution goes here  */
11
12    if (isTeenager) {
13       cout << "Teen" << endl;
14    }
```

```
15    else {
```

Run

**CHALLENGE ACTIVITY** 3.12.2: 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)



Learn how our autograder works

539740.3879454.qx3zqy7

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

int main() {
   bool isRed;
   bool isBalloon;

   cin >> isRed;
   cin >> isBalloon;

   /* Your solution goes here  */

   return 0;
}
```

Run

# 3.13 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** 3.13.1: 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 3.13.1: String equality example.

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

int main() {
   string userWord;

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

   if (userWord == "USA") {
      cout << "United States of
America";
   }
   else {
      cout << userWord;
   }
   cout << endl;

   return 0;
}
```

```
Enter a word: Sally
Sally

...

Enter a word: USA
United States of
America

...

Enter a word: usa
usa
```

**PARTICIPATION ACTIVITY** 3.13.2: 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"

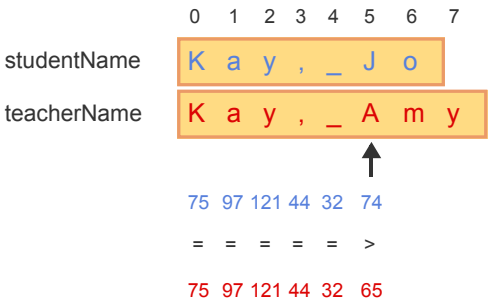
- True
- False

## 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** 3.13.3: String comparison.




### Animation content:

Begin static figure:
An example of string comparison is given with two different strings. The first string, a variable called studentName, is "Kay,_Jo". The second string, a variable called teacherName, is "Kay,_Amy". Below the strings is a walkthrough of the string comparison character by character. The string comparison uses ASCII values to check equality. The first 5 characters (characters K a y , _ at indices 0 to 4) are the same, and are represented by the following ASCII values: 75, 97, 121, 44, 32. An arrow showing where we are in comparing the strings stops at the sixth character. In the first string, the character is "J" while in the second string, the character is "A". The ASCII value of "J" is 74 and the ASCII value of "A" is 65, therefore "Kay,_Jo" is greater than "Kay,_Amy".

Step 1: A string comparison compares each character.
The first five characters of studentName and teacherName are the same.
Each character in studentName and teacherName are converted to ASCII values and compared.
The first 5 characters in both studentName and teacherName are "Kay, Jo", and are converted to 75, 97, 121, 44, 32. These values are equal for both strings.
Step 2: 'J' is greater than 'A', so studentName is greater than teacherName.
"J" is converted to 74 in ASCII, and "A" is converted to 65. "74 > 65" is shown..



### 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** 3.13.4: 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** 3.13.5: 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** 3.13.1: String comparison: Detect word. Full screen

539740.3879454.qx3zqy7

Organize the code to create an if-else statement that prints "Goodbye" if userString is "Quit", else prints "Hello".

Note: This activity includes distractors. Not all code statements on the left will be used in the final solution.

How to use this tool

**Unused**

```
cout << "Hello";
cout << "Hello" << endl;
else if {
else {
if (userString = "Quit") {
if (userString == "Quit") {
}
cout << "Goodbye";
cout << "Goodbye" << endl;
}
```

**main.cpp** Load default templ

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

int main() {
   string userString;

   cin >> userString;

   return 0;
}
```



Check

**CHALLENGE ACTIVITY** 3.13.2: 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
```

Learn how our autograder works

539740.3879454.qx3zqy7

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

Run

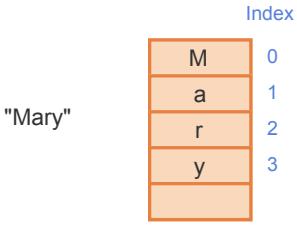
# 3.14 String access operations



## 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** 3.14.1: A string's characters each has an index, starting with 0.




### Animation content:

Static figure:
The word Mary is next to a sequence of memory locations labeled with indices 0 to 3. M is placed in the block at index 0, a at 1, r at 2, and y at 3.

### 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** 3.14.2: 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 3.14.1: 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**

3.14.3: 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 3.14.2: 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 3.14.4: 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 3.14.3: 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** 3.14.5: 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** 3.14.6: 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** 3.14.7: 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

## Common errors

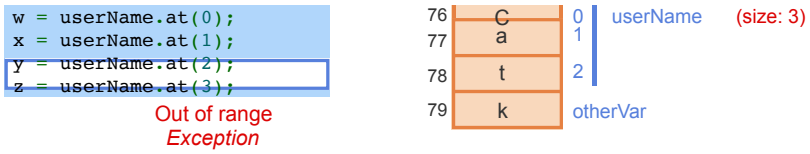
*A common error is to access an invalid string index, especially exactly one larger than the largest index.* Given userText with size 8, the range of valid indices are 0 … 7; accessing with index 8 is an error.



**PARTICIPATION ACTIVITY** 3.14.8: Common error: Out-of-range access yields an exception.

```
userName = "Cat";
```

75 …




### Animation content:



Static figure:
Begin C++ code:
userName = "Cat";
w = userName.at(0);
x = userName.at(1);
y = userName.at(2);
z = userName.at(3);
End C++ code.

A sequence of 5 memory blocks is next to the code. The memory blocks start at memory location 75 and end at 79. A C is stored at memory location 76 and is associated with the index 0 of the string variable userName. An a is stored at memory location 77 and is associated with index 1. A t is stored at memory location 78 and is associated with index 2. The size of userName is 3. A k is stored at memory location 79 and is associated with the variable otherVar.

Step1: 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.
Step 2: Accesses to indices 0, 1, 2 are fine. But accessing index 3 is an error. Trying to access index 3 results in Out of range 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** 3.14.9: 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** 3.14.1: String access operations.

539740.3879454.qx3zqy7

Start

Given string userStr on one line and integers idx1 and idx2 on a second line, change the character at index idx1 of userStr to character at index idx2.

Ex: If the input is:

```
garlic
2 0
```

then the output is:

```
gaglic
```

Note: Assume the length of string userStr is greater than or equal to both idx1 and idx2.

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main() {
6     string userStr;
7     int idx1;
8     int idx2;
9
10    getline(cin, userStr);
11    cin >> idx1;
12    cin >> idx2;
13
14    /* Your code goes here */
15
```

1 2 3 4



Check Next level

# 3.15 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 3.15.1: Character functions return values.



| | | | | | | |
|---|---|---|---|---|---|---|
| ***isalpha***(c) | true if alphabetic: a-z or A-Z | `isalpha('x')`<br>`// true`<br>`isalpha('6')`<br>`// false`<br>`isalpha('!')`<br>`// false` | | ***toupper***(c) | Uppercase version | `letter = toupper('a')`<br>`// A`<br>`letter = toupper('A')`<br>`// A`<br>`letter = toupper('3')`<br>`// 3` |
| ***isdigit***(c) | true if digit: 0-9. | `isdigit('x')`<br>`// false`<br>`isdigit('6')`<br>`// true` | | ***tolower***(c) | Lowercase version | `letter = tolower('A')`<br>`// a`<br>`letter = tolower('a')`<br>`// a`<br>`letter = tolower('3')`<br>`// 3` |
| ***isspace***(c) | true if whitespace. | `isspace(' ')`<br>`// true`<br>`isspace('\n')`<br>`// true`<br>`isspace('x')`<br>`// 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 3.15.1: 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** 3.15.1: 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** 3.15.1: Character operations.

539740.3879454.qx3zqy7

Start

A 2-character string, inputString, is read from input. Replace each character in inputString that is *not* alphabetic with '#'. Otherwise, inputString is not changed.

Ex: If the input is `0c`, then the output is:

```
#c
```

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

int main() {
   string inputString;

   getline(cin, inputString);

   /* Your code goes here */

   cout << inputString << endl;

   return 0;
```

1 2 3



Check Next level

# 3.16 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 3.16.1: 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).<br><br>find(item, indx) starts at index indx. | `// userText is "Help me!"`<br>`userText.find('p')  // Returns 3`<br>`userText.find('e')  // Returns 1 (first occurrence of e only)`<br>`userText.find('z')  // Returns string::npos`<br>`userText.find("me")  // Returns 5`<br>`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"`<br>`userText.substr(0, 7)  // Returns "http://"`<br>`userText.substr(13, 4)  // Returns ".com"`<br>`userText.substr(userText.size() - 4, 4)  // Last 4: ".com"` |

Figure 3.16.1: 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** 3.16.1: 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 3.16.2: 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"`<br>`userText.push_back('?'); // Now "Hello?"`<br>`userText.size();  // Returns 6` |
| insert() | ***insert***(indx, subStr) Inserts string subStr starting at index indx. | `// userText is "Goodbye"`<br>`userText.insert(0, "Well "); // Now "Well Goodbye"`<br><br>`// userText is "Goodbye"`<br>`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"`<br>`userText.replace(9, 4, "a plethora of");`<br>`// Now "You have a plethora of gifts"` |
| str1 + str2 | Returns a new string that is a copy of str1 with str2 appended.<br><br>If one of str1, str2 is a string, the other may | `// userText is "A B"`<br>`myString = userText + " C D";`<br>`// myString is "A B C D"`<br>`myString = myString + '!';`<br>`// myString now "A B C D!"`<br>`myString = myString + userText;`<br>`// myString now "A B C D!A B"` |

| | | |
|---|---|---|
| | be a character (or character array). | |

Figure 3.16.2: 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 "@$#"
   if (greetingText.find("Darn") != string::npos) { //
Found
      itemIndex = greetingText.find("Darn");
      greetingText.replace(itemIndex, 4, "@#$");
   }
   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** 3.16.2: 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** 3.16.1: Combining strings.

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

```
Barry Allen
```

Learn how our autograder works

539740.3879454.qx3zqy7

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

Run

**CHALLENGE ACTIVITY** 3.16.2: Basic string manipulation.

539740.3879454.qx3zqy7

Start

Given string userInput on one line and character suffix on a second line, append suffix to userInput.

Ex: If the input is:

```
Fuzzy bear
!
```

then the output is:

```
Fuzzy bear!
```



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

int main() {
   string userInput;
   char suffix;

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

   /* Your code goes here */

   cout << userInput << endl;

```

1 2 3 4

Check Next level

# 3.17 Conditional expressions

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

**PARTICIPATION ACTIVITY** 3.17.1: Conditional expression.



```
if (condition) {
  myVar = expr1;
}
else {
  myVar = expr2;
}
```

```
myVar = (condition) ? expr1 : expr2;
```

### Animation content:

There are two code snippets. On the left the code snippet is:

```
if (condition) {
  myVar = expr1;
}
else {
  myVar = expr2;
}
```



The second code snippet is equivalent to the first and is on the right. It reads:
myVar = (condition) ? expr1:expr2

### 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** 3.17.2: 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 | 3.17.1: Conditional expression: Print negative or positive. Full screen

539740.3879454.qx3zqy7



Drag and drop the conditional expression that evaluates to string "negative" if userVal is less than 0, and "non-negative" other into main.cpp.

Ex: If userVal is -9, output is:

```
-9 is negative.
```

How to use this tool

**Unused**

**main.cpp** Load default templ

```
(userVal < 0) ? "negative" : "non-negative";
condStr = (userVal < 0) ? "negative" : "non-negative";
condStr = (userVal < 0) ? "non-negative" : "negative";
condStr = (userVal > 0) ? "negative" : "non-negative";
```

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

int main() {
   string condStr;
   int userVal;

   cin >> userVal;

   cout << userVal << " is " << condStr << "." << e

   return 0;
}
```



Check

**CHALLENGE ACTIVITY** 3.17.2: 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 = ...".

Learn how our autograder works

539740.3879454.qx3zqy7

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     int numUsers;
6     int updateDirection;
7
8     cin >> numUsers;
9     cin >> updateDirection;
10
11    /* Your solution goes here  */
12
13    cout << "New value is: " << numUsers << endl;
14
15    return 0;
```



Run

**CHALLENGE ACTIVITY** 3.17.3: Conditional expressions: Enter the output of the code.

539740.3879454.qx3zqy7

Start

Type the program's output

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

int main() {
   int myNumber;
   int yourNumber;

   myNumber = 9;
   yourNumber = (myNumber >= 1) ? 6 : 3;

   cout << yourNumber << endl;

   return 0;
}
```

6



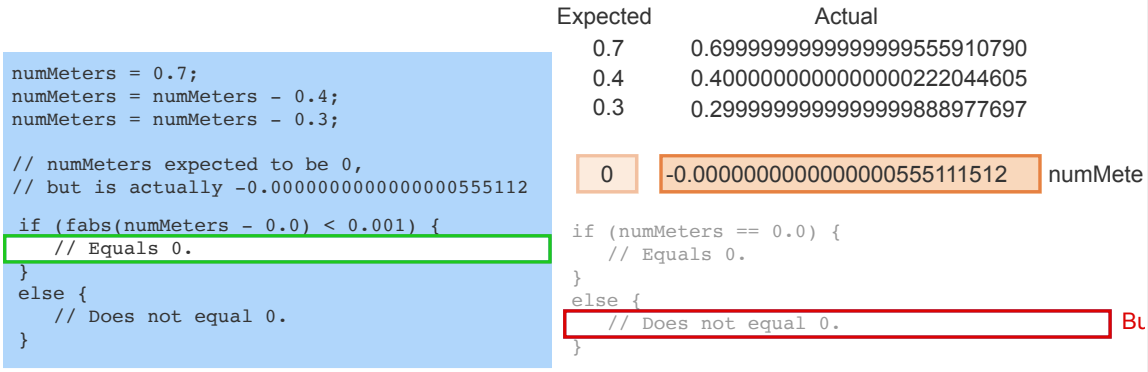
1 2 3

Check Next

# 3.18 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** 3.18.1: Floating-point comparisons.




**Animation content:**

Static figure:
Floating-point numbers' expected and actual values (as stored in memory) are shown. 0 is stored as -0.00000000000000000555112

A bug occurs if the comparison code uses ==
Begin code:

```
if (numMeters == 0.0) {
  // Equals 0.
}
else {
```

```
  // Does not equal 0.
}
End code.

'close enough' comparison reads:
Begin code:
numMeters = 0.7;
numMeters = numMeters - 0.4;
numMeters = numMeters - 0.3;

// numMeters expected to be 0,
// but is actually -0.00000000000000000555112
if (fabs(numMeters - 0.0) < 0.001) {
  // Equals 0.
}
else {
  // Does not equal 0.
}
End code.
```



### 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** 3.18.2: 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** 3.18.3: 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** 3.18.4: 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 3.18.1: 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** 3.18.5: 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 3.18.2: 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
0.6999999999999999555911079
sampleValue4 is 0
sampleValue5 is 0.25
```



**PARTICIPATION ACTIVITY** 3.18.6: Inexact representation of floating-point values.

Enter a decimal value: _____ C

| 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 0 0 0 0 0 0 0 |

**PARTICIPATION ACTIVITY** 3.18.7: 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** 3.18.1: Floating-point comparison: Print Equal or Not equal. Full screen



539740.3879454.qx3zqy7

Drag and drop the if statement that will cause the following code to print "Equal" if the value of sensorReading is "close enou targetValue. Otherwise, print "Not equal".

Ex: If targetValue is 0.3333 and sensorReading is (1.0/3.0), the output is:

```
Equal
```

How to use this tool

**Unused**

```
if (fabs(targetValue - sensorReading) < 0.0001) {
if (sensorReading == targetValue) {
if (fabs(targetValue - sensorReading) == 0.0001) {
```

**main.cpp** Load default templ

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

int main() {
   double targetValue;
   double sensorReadingNumerator;
   double sensorReadingDenominator;
   double sensorReading;

   cin >> targetValue;
   cin >> sensorReadingNumerator;
   cin >> sensorReadingDenominator;

   sensorReading = sensorReadingNumerator / sensorR

      cout << "Equal" << endl;
   }
```

Check

# 3.19 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.

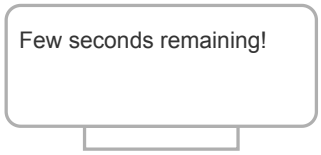
**PARTICIPATION ACTIVITY** 3.19.1: Short circuit evaluation: Logical AND.

```
// Read minutes and seconds
```

operand1 operand2 result

```
if ((minutes < 1) && (seconds < 10)) {
  // Print "Few seconds remaining!"
}
else {
  // Print "Take your time."
}
...
```

| | | | |
|---|---|---|---|
| minutes: 1<br>seconds: 5 | (1 < 1)<br>false | - | false |
| minutes: 0<br>seconds: 5 | (0 < 1)<br>true | (5 < 10)<br>true | true |






### Animation content:

Static figure:
Begin code:
// Read minutes and seconds
if ((minutes < 1) && (seconds < 10)) {
  // Print "Few seconds remaining!"
}
else {
  // Print "Take your time."
}
...
End code.

Two cases with different values for the variable minutes and seconds exist.
In the first case, minutes equals 1 and seconds equals 5. The first operand evaluates to 1 < 1 that is false, so the result is false.
In the second case, minutes equals 0 and seconds equals 5. The first operand evaluates to 0 < 1 that is true. The second operand evaluates to 5 < 10 that is also true. The result is true and the computer screen at the bottom displays the output "Few seconds remaining!"

### 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 3.19.1: 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** 3.19.2: 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)



# 3.20 C++ example: Salary calculation with branches

## zyDE 3.20.1: Calculate salary: Calculate overtime using branches.

The following program calculates weekly salary and assumes work-hours-per-week limit of 40.

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.

Load default template...

```
#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;

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



10 42

Run

## zyDE 3.20.2: 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.



Note: The calculation is inaccurate to how taxes are formally assessed and is a simplification for educational purposes only.

Load default template...

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

int main() {
   int annualSalary;
   double taxRate;
   int taxToPay;

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

   // Determine the tax rate from the annual salary
   // FIXME: Write code to address the challenge ques
   if (annualSalary <= 20000) {
      taxRate = 0.10;
```



120000

Run

# 3.21 C++ example: Search for name using branches

## zyDE 3.21.1: 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 (ICANN: 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.

Load default template...

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

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

   coreGtld1 = ".com";
```



.info

Run

Below is a solution to the above problem.

## zyDE 3.21.2: Search for name using branches (solution).

Load default template...

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

int main() {
   string inputName;
   string searchName;
   string coreGtld1;
   string coreGtld2;
   string coreGtld3;
   string coreGtld4;
   bool isCoreGtld;

   coreGtld1 = ".com";
```



.info

Run

# 3.22 LAB: Smallest number

Write a program whose inputs are three integers, and whose output is the smallest of the three values.

Ex: If the input is:

```
7 15 3
```

the output is:

```
3
```



LAB ACTIVITY | 3.22.1: LAB: Smallest number | 0 / 10



main.cpp Load default template...

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

int main() {

   /* Type your code here. */

   return 0;
```

```
9 }
```

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) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

# 3.23 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, and service the primary highway indicated by the rightmost two digits. Thus, I-405 services I-5, and I-290 services I-90. Note: 200 is not a valid auxiliary highway because 00 is not a valid primary highway number.

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** | 3.23.1: LAB: Interstate highway numbers | 0 / 10

main.cpp — Load default template...

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

int main() {
   int highwayNumber;

   cin >> highwayNumber;

   /* Type your code here. */

   return 0;
}
```

**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) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```



# 3.24 LAB: Exact change

Write a program with total change amount as an integer input, and output the change using the fewest coins, one coin type per line. The coin types are Dollars, Quarters, Dimes, Nickels, and Pennies. Use singular and plural coin names as appropriate, like 1 Penny vs. 2 Pennies.

Ex: If the input is:

```
0
```

(or less than 0), the output is:

```
No change
```

Ex: If the input is:

```
45
```

the output is:

```
1 Quarter
2 Dimes
```



LAB ACTIVITY | 3.24.1: LAB: Exact change 0 / 10

main.cpp Load default template...

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

int main() {

   /* Type your code here. */

   return 0;
}
```

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) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

# 3.25 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; therefore, both 1700 and 1800 are not leap years

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 | 3.25.1: LAB: Leap year | 0 / 10

main.cpp — Load default template...



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

int main() {
   int inputYear;
   bool isLeapYear = false;

   cin >> inputYear;

   /* Type your code here. */

   return 0;
}
```

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) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.



# 3.26 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 | 3.26.1: LAB: Name format | 0 / 10

main.cpp — Load default template...

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

int main() {

   /* Type your code here. */

   return 0;
}
```

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) → Output (shown below)

Program output displayed here



Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

# 3.27 LAB: Login name

Write a program that creates a login name for a user, given the user's first name, last name, and a four-digit integer as input. Output the login name, which is made up of the first six letters of the first name, followed by the first letter of the last name, an underscore (_), and then the last digit of the number (use the % operator). If the first name has less than six letters, then use all letters of the first name.

Ex: If the input is:

```
Michael Jordan 1991
```

the output is:

```
Your login name: MichaeJ_1
```

Ex: If the input is:

```
Nicole Smith 2024
```

the output is:

```
Your login name: NicoleS_4
```



LAB ACTIVITY | 3.27.1: LAB: Login name 0 / 10

main.cpp Load default template...



```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main() {
6     string login;
7     string first;
8     string last;
9     int number;
10
```