4.1 LAB: Varied amount of input data

Statistics are often calculated with varying amounts of input data. Write a program that takes any number of non-negative integers as input, and outputs the max and average. A negative integer ends the input and is not included in the statistics. Assume the input contains at least one non-negative integer.

Output each floating-point value with two digits after the decimal point, which can be achieved by executing `cout << fixed << setprecision(2);` once before all other cout statements.

Ex: When the input is:

```
15 20 0 3 -1
```

the output is:

```
20 9.50
```


LAB ACTIVITY | 4.1.1: LAB: Varied amount of input data | 0 / 10

main.cpp — Load default template...

```
#include <iostream>
#include <iomanip>
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?

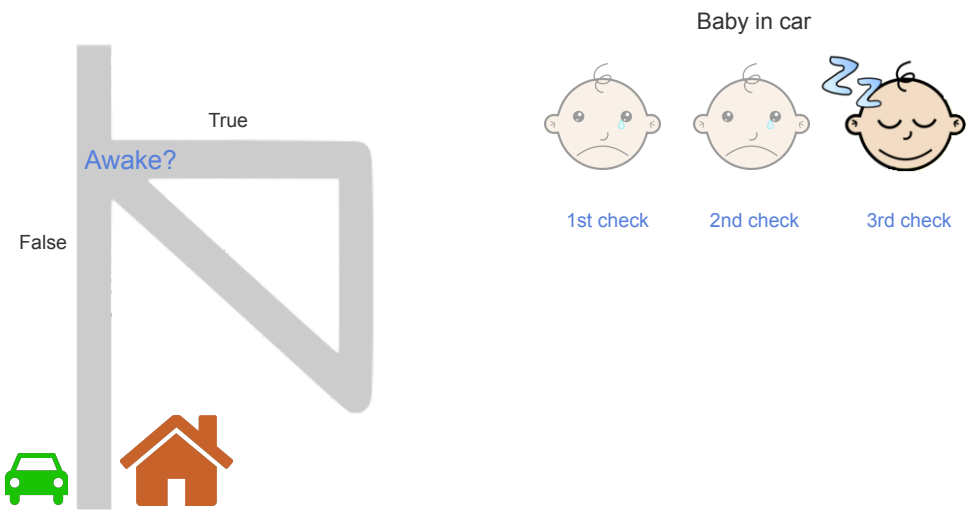
```
History of your effort will appear here once you begin working
on this zyLab.
```


4.2 Loops (general)

Loop concept

People who have children may be familiar with looping around the block until a baby falls asleep.

PARTICIPATION ACTIVITY 4.2.1: Loop concept: Driving a baby around the block.



Animation content:

Static figure:
The parent's car arrives at a fork in the road and is presented with two road choices to choose from: road True or road False. The first road, True, takes the parents back at the fork again after a loop. The second road, False, takes the parents directly home. At the fork is the question, Awake?. If the baby is awake, the answer to the question is true, and the parents would choose the looping road, True. Otherwise, the baby is asleep, the answer to the question is false, and the parents would drive straight home on the road False.

Step 1: Parents may be familiar with this scenario: Driving home, baby is awake. Parents circle the block, hoping the baby will fall asleep. The parent's car arrives at the fork: road True, the loop, or road False, go home. The baby is initially awake, the answer is true, and the parents loop back at the fork.

Step 2: After first loop, baby is still awake, so parents loop again.

Step 3: After second loop, baby is asleep, so parents head home for a peaceful evening.

Animation captions:

1. Parents may be familiar with this scenario: Driving home, baby is awake. Parents circle the block, hoping the baby will fall asleep.
2. After first loop, baby is still awake, so parents loop again.
3. After second loop, baby is asleep, so parents head home for a peaceful evening.

PARTICIPATION ACTIVITY 4.2.2: Loop concept.

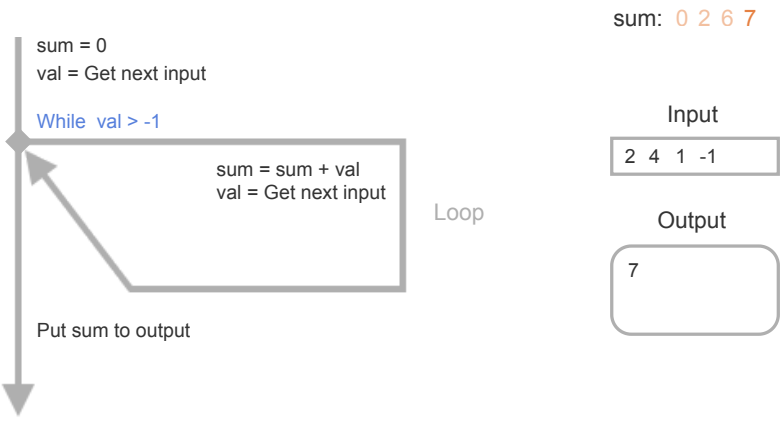
Consider the example above.

1) When the parents first checked, was the baby awake?
   - Yes
   - No
2) After the first loop, was the baby awake?
   - Yes
   - No
3) After the second loop, was the baby awake?
   - Yes
   - No
4) How many loops around the block did the parents make?
   - 2
   - 3
5) Where was the decision point for whether to loop: At the top of the street or bottom?
   - Top
   - Bottom

Loop basics

A ***loop*** is a program construct that repeatedly executes the loop's statements (known as the ***loop body***) while the loop's expression is true; when the expression is false, execution proceeds past the loop. Each time through a loop's statements is called an ***iteration***.

PARTICIPATION ACTIVITY 4.2.3: A simple loop: Summing the input values.



Animation content:

Static Figure:
A diagram depicts a straight line tangent to a circle, labeled Loop, as well as an input console and output console. A small dot travels South along the straight line until it reaches the point of tangency. Then, the dot travels around the circle, or loop. After one complete loop around the circle, the dot reaches the point of tangency again, and continues traveling South on the straight line. Various code statements are placed along the straight line and the circle. A code statement is executed if the dot travels by.

Step 1: A loop is like a branch, but jumping back to the expression when done. Thus, the loop's statements may execute multiple times, before execution proceeds past the loop. A dot travels along a straight path until arriving at a fork. Two paths are available to choose from: the first path branches off of the straight path and loops back to the fork again, and the second path would allow the dot to continue straight.

The dot's initial location is at the beginning of the straight path, before the fork has been reached. The input console contains one line of input: 2 4 1 -1. The output console is initially empty. As the dot travels, arriving at various statements, the statements are immediately executed. The path the dot travels matches the order of the tables below.

Step 2: This program gets an input value. If the value > -1, the program adds the value to a sum, gets another input, and repeats. val is 2, so the loop's statements execute, making sum 2.

Dot's current location	Statement	Execution
Before the loop, first statement	sum = 0	sum = 0
Before the loop, second statement	val = Get next input	val = 2
At the fork, or loop access point	While val > -1	returns true since 2 is greater than -1, so the loop is entered
Within the loop, first statement	sum = sum + val	sum = 0 + 2, so sum = 2

Step 3: The loop statement is ended by getting the next input, which is 4. The loop's expression 4 > -1 is true, so the loop's statements execute again, making sum 2 + 4 or 6.

Dot's current location	Statement	Execution
Within the loop, second statement	val = Get next input	val = 4
At the fork, or loop access point	While val > -1	returns true since 4 is greater than -1, so the loop is entered
Within the loop, first statement	sum = sum + val	sum = 2 + 4, so sum = 6

Step 4: The loop's statements got the next input of 1. The loop's expression 1 > -1 is true, so the loop's statements execute a third time, making sum 6 + 1 or 7.

Dot's current location	Statement	Execution
Within the loop, second statement	val = Get next input	val = 1
At the fork, or loop access point	While val > -1	returns true since 1 is greater than -1, so the loop is entered
Within the loop, first statement	sum = sum + val	sum = 6 + 1, so sum = 7

Step 5: The next input is -1. This time, -1 > -1 is false, so the loop is not entered. Instead, execution proceeds past the loop, where a statement puts sum, which is 7, to the output.

Dot's current location	Statement	Execution
Within the loop, second statement	val = Get next input	val = -1
At the fork, or loop access point	While val > -1	returns false since -1 is not greater than -1, so the loop terminates
After the loop	Put sum to output	Output console contains 7

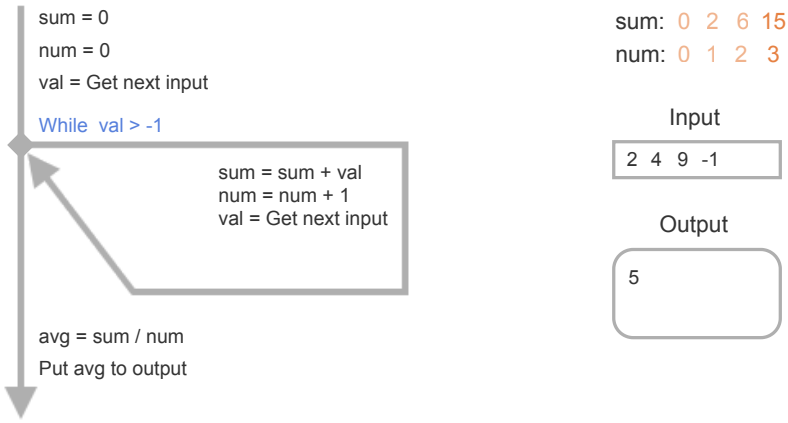
Animation captions:

1. A loop is like a branch, but the loop jumps back to the expression when done. Thus, the loop's statements may execute multiple times before execution proceeds past the loop.
2. This program receives an input value. If the value > -1, the program adds the value to a sum, receives another input, and repeats. val is 2, so the loop's statements execute, making sum 2.
3. The loop's statements ended by receiving the next input, which is 4. The loop's expression 4 > -1 is true, so the loop's statements execute again, making sum 2 + 4 or 6.
4. The loop's statements receive the next input of 1. The loop's expression 1 > -1 is true, so the loop's statements execute a third time, making sum 6 + 1 or 7.
5. The next input is -1. This time, -1 > -1 is false, so the loop is not entered. Instead, execution proceeds past the loop, where a statement puts sum, which is 7, to the output.

Loop example: Computing an average

A loop can be used to compute the average of a list of numbers.

PARTICIPATION ACTIVITY 4.2.4: Loop example: Computing an average.



Animation content:

A dot travels along a straight path until arriving at a fork. Two paths are available to choose from: the first path branches off of the straight path and loops back to the fork again, and the second path would allow the dot to continue straight.

The dot's initial location is at the beginning of the straight path, before the fork has been reached. The input console contains one line of input: 2 4 9 -1. The output console is initially empty. As the dot travels, arriving at various statements, the statements are immediately executed. The path the dot travels matches the order of the tables below.

Step 1: The program computes an average of a list of numbers (a negative ends the list). The first input is 2, so the loop is entered. Sum becomes 2, and num is incremented to 1.

Dot's current location	Statement	Execution
Before the loop, first statement	sum = 0	sum = 0
Before the loop, second statement	num = 0	num = 0
Before the loop, third statement	val = Get next input	val = 2
At the fork, or loop access point	While val > -1	returns true since 2 is greater than -1, so the loop is entered
Within the loop, first statement	sum = sum + val	sum = 0 + 2, so sum = 2
Within the loop, second statement	num = num + 1	num = 0 + 1, so num = 1

Step 2: The next input is 4. The loop is entered, so sum becomes 2 + 4 or 6, and num is incremented to 2.

Dot's current location	Statement	Execution
Within the loop, third statement	val = Get next input	val = 4
At the fork, or loop access point	While val > -1	returns true since 4 is greater than -1, so the loop is entered
Within the loop, first statement	sum = sum + val	sum = 2 + 4, so sum = 6
Within the loop, second statement	num = num + 1	num = 1 + 1, so num = 2

Step 3: The next input is 9, so the loop is entered. Sum becomes 6 + 9 or 15, and num is incremented to 3.

Dot's current location	Statement	Execution
Within the loop, third statement	val = Get next input	val = 9
At the fork, or loop access point	While val > -1	returns true since 9 is greater than -1, so the loop is entered
Within the loop, first statement	sum = sum + val	sum = 6 + 9, so sum = 15
Within the loop, second statement	num = num + 1	num = 2 + 1, so num = 3

Step 4: The next input is -1, so the loop is not entered. 15 / 3 or 5 is output.

Dot's current location	Statement	Execution
Within the loop, second statement	val = Get next input	val = -1
At the fork, or loop access point	While val > -1	returns false since -1 is not greater than -1, so the loop terminates
After the loop, first statement	avg = sum / num	avg = 15 / 3, so avg = 5
After the loop	Put avg to output	Output console contains 5

Animation captions:

1. The program computes an average of a list of numbers (a negative ends the list). The first input is 2, so the loop is entered. Sum becomes 2, and num is incremented to 1.

2. The next input is 4. The loop is entered, so sum becomes 2 + 4 or 6, and num is incremented to 2.
3. The next input is 9, so the loop is entered. Sum becomes 6 + 9 or 15, and num is incremented to 3.
4. The next input is -1, so the loop is not entered. 15 / 3 or 5 is output.

PARTICIPATION ACTIVITY 4.2.5: Loop example: Average.

Consider the computing an average example above.

1) In the example above, the first value received from input was 2. That caused the loop body to be _____.
   - executed
   - not executed

2) At the end of the loop body, the _____.
   - next input is received
   - loop is exited
   - average is computed

3) With what value was sum initialized?
   - -1
   - 0

4) Each time through the loop, the sum variable is increased by _____.
   - 0
   - 1
   - the current input value

5) What was variable num's value after the loop was done iterating?
   - 1
   - 2
   - 3

6) Before the loop, the first input value is received. If that input was negative (unlike the data in the example above), the loop's body would _____.
   - be executed
   - not be executed

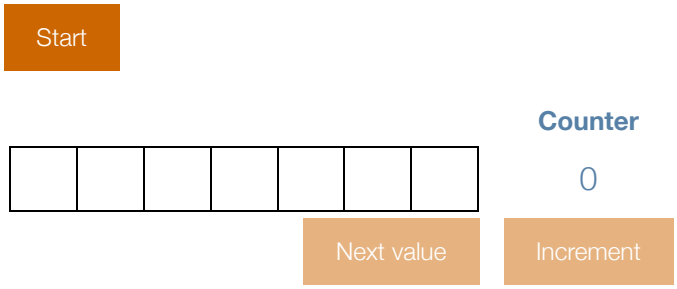
Example: Counting specific values in a list

Programs execute one statement at a time. Thus, using a loop to examine a list of values one value at a time and updating variables along the way, as in the above examples, is a common programming task.

Below is a task to help a person get accustomed to examining a list of values one value at a time. The task asks a person to count the number of negative values, incrementing a variable to keep count.

PARTICIPATION ACTIVITY 4.2.6: Counting negative values in a list of values.

Click "Increment" if a negative value is seen.



PARTICIPATION ACTIVITY 4.2.7: Counting negative values.

Complete the program such that variable count ends having the number of negative values in an input list of values (the list ends with 0). So if the input is -1 -5 9 3 0, then count should end with 2.

```
count = 0
val = Get next input

While val is not 0
   If __(A)__
      __(B)__

   val = Get next input
```

1) What should expression (A) be?

- val > 0
- val < 0
- val is 0

2) What should statement (B) be?

- val = val + 1
- count = count + 1
- count = val

3) If the input value is 0, does the loop body execute?

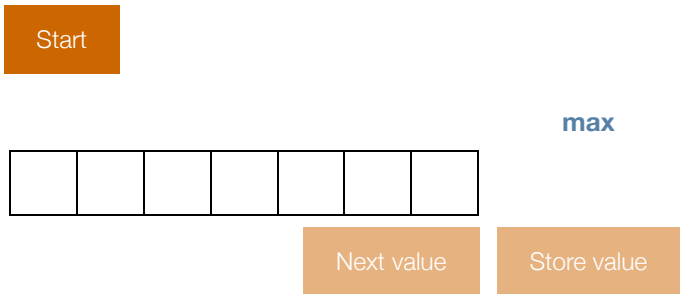
- Yes
- No

Example: Finding the max value

Examining items one at a time and updating a variable can achieve some interesting computations. The task below is to find the maximum value in a list of positive values. A variable stores the max value seen so far. This variable is initialized with the first input value. Each following input value is compared with the stored max value, and if greater, the input value replaces the max value.

PARTICIPATION ACTIVITY 4.2.8: Find the maximum value in the list of values.

Click "Store value" to initialize max with the first input value, and then click "Store value" if a new maximum value is seen.



Time - Best time -
Clear best

PARTICIPATION ACTIVITY 4.2.9: Determining the max value.

Complete the program such that variable max ends up having the maximum value in an input list of positive values (the list ends with 0). So if the input is 22 5 99 3 0, then max should end as 99.

```
val = Get next input
max = val
val = Get next input

While val is not 0
   If __(A)__
      __(B)__

   val = Get next input
```

1) What should expression (A) be?

- max > 0
- max > val
- val > max

2) What should statement (B) be?

- max = val
- val = max
- max = max + 1

3) Does the final value of max depend on the order of inputs? In particular, would max be different for inputs 22 5 99 3 0 versus inputs 99 3 5 22 0?

- Yes
- No

4) For inputs 5 10 7 20 8 0, with what values will max be assigned?

- 20
- 5, 10, 20
- 5, 10, 7, 20

4.3 While loops

While loop: Basics

A ***while loop*** is a program construct that repeatedly executes a list of sub-statements (known as the ***loop body***) while the loop's expression evaluates to true. Each execution of the loop body is called an ***iteration***. Once entering the loop body, execution continues to the body's end, even if the expression would become false midway through.

Construct 4.3.1: While loop.

```
while (expression) { // Loop expression
    // Loop body: Executes if expression evaluated to true
    // After body, execution jumps back to the "while"
}
// Statements that execute after the expression evaluates to
false
```

PARTICIPATION ACTIVITY 4.3.1: While loop.

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

int main() {
  int currPower;
  char userChar;

  currPower = 2;
  userChar = 'y';

  while (userChar == 'y') {
     cout << currPower << endl;
     currPower = currPower * 2;
     cin >> userChar;
  }

  cout << "Done" << endl;
```

Input

y y n

Output

```
2
4
8
Done
```

```
   return 0;
}
```

Animation content:

Static figure:
Begin C++ code:
#include <iostream>
using namespace std;

int main() {
  int currPower;
  char userChar;

  currPower = 2;
  userChar = 'y';
 `
  while (userChar == 'y') {
     cout << currPower << endl;
     currPower = currPower * 2;
     cin >> userChar;
 }
  cout << "Done" << endl;

  return 0;
}
End C++ code.

The input code y y n is displayed on one line.

The output code
2
4
8
Done
is displayed on four lines. Each output is on its own line.

The initial userChar value, 'y', has the corresponding output 2.
The input 'y' has the corresponding output 4.
The input 'y' has the corresponding output 8.
The input 'n' has the corresponding output "Done".

Animation captions:

1. When encountered, a while loop's expression is evaluated. If true, the loop's body is entered. Here, userChar was initialized with 'y', so userChar == 'y' is true.
2. Thus, the loop body is executed, which outputs currPower's current value of 2, doubles currPower, and gets the next input.
3. Execution jumps back to the while part. userChar is 'y' (the first input), so userChar == 'y' is true, and the loop body executes (again), outputting 4.
4. userChar is 'y' (the second user input), so userChar == 'y' is true, and the loop body executes (a third time), outputting 8.

5. userChar is now 'n', so userChar == 'y' is false. Thus, execution jumps to after the loop, which outputs "Done".

PARTICIPATION ACTIVITY 4.3.2: While loops: Number of iterations.

For the following code, indicate how many times the loop body will execute for the indicated input values.


```
int userNum = 3;

while (userNum > 0) {
   // Do something
   userNum = // Get input into userNum
}
```

1) Input: 5 -1

Check Show answer

2) Input: 2 1 0

Check Show answer

3) Input: -1 5 4

Check Show answer

Basic while loop example

The following Celsius to Fahrenheit example shows the common pattern of getting user input at the end of each loop iteration to determine whether to continue looping.

Figure 4.3.1: While loop example: Celsius to Fahrenheit.

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

int main() {
   double celsiusValue;
   double fahrenheitValue;
   char userChar;

   celsiusValue = 0.0;
   userChar = 'y';

   while (userChar == 'y') {
      fahrenheitValue = (celsiusValue * 9.0 /
5.0) + 32.0;

      cout << celsiusValue << " C is ";
      cout << fahrenheitValue << " F" << endl;

      cout << "Type y to continue, any other to
quit: ";
      cin >> userChar;

      celsiusValue = celsiusValue + 5;
      cout << endl;
   }

   cout << "Goodbye." << endl;

   return 0;
}
```

```
0 C is 32 F
Type y to continue, any other to
quit: y

5 C is 41 F
Type y to continue, any other to
quit: y

10 C is 50 F
Type y to continue, any other to
quit: y

15 C is 59 F
Type y to continue, any other to
quit: y

20 C is 68 F
Type y to continue, any other to
quit: y

25 C is 77 F
Type y to continue, any other to
quit: y

30 C is 86 F
Type y to continue, any other to
quit: y

35 C is 95 F
Type y to continue, any other to
quit: y

40 C is 104 F
Type y to continue, any other to
quit: q

Goodbye.
```


PARTICIPATION ACTIVITY 4.3.3: While loop example: Celsius to Fahrenheit.

Consider the example above.

1) The loop will always iterate at least once.
   - True
   - False

2) For the user inputs shown, how many times did the loop iterate?
   - 9
   - 10

3) Each iteration adds ____ to celsiusValue.
   - 1
   - 5

4) If the user's first input is 'n', how many times will the loop iterate?
   - 1
   - 2

Getting input before a loop

The above examples got user input into a variable only at the end of the loop body. The examples assigned that variable an initial value that always caused the loop body to execute the first time. Another common pattern gets that initial value from user input as well, thus getting input in two places: before the loop, and at the loop body's end.

Figure 4.3.2: Common pattern: Getting input before and at end of loop.


```
// Get input into userChar

while (userChar == 'y') {
   // Do something ...
   // Get input into
userChar
}
```

PARTICIPATION ACTIVITY 4.3.4: While loops: Number of iterations, with input gotten before the loop.

For the following code, indicate how many times the loop body will execute for the indicated input values.

```
int userNum = //Get input into userNum

while (userNum > 0) {
   // Do something ...
   userNum = //Get input into userNum
}
```

1) Input: 5 -1

Check Show answer

2) Input: 2 1 0

Check Show answer

3) Input: -1 5 4

Check Show answer

Loop expressions

Various kinds of expressions are found in while loop expressions. For example, sometimes a loop is executed as long as a value is greater than another value, or less than another value. Sometimes a loop is executed as long as a value is NOT equal to another value.

Below is an example with a relational operator in the loop expression.

PARTICIPATION ACTIVITY 4.3.5: While loop using a relational operator in the loop expression.

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

int main() {
  int userNum;

  cin >> userNum;

  while (userNum > 0) {
     cout << userNum % 10 << endl;
     userNum = userNum / 10;
  }

  return 0;
}
```

Iteration	userNum	Output
	902	
1		2
	90	
2		0
	9	
3		9
	0	

Animation content:

Static figure:
Begin C++ code:
#include <iostream>
using namespace std;

int main() {
 int userNum;

 cin >> userNum;

 while (userNum > 0) {
  cout << userNum % 10 << endl;
  userNum = userNum / 10;
 }

 return 0;
}
End C++ code.

Iteration	userNum	Output
	902	
1		2
	90	
2		0
	9	
3		9
	0	

Animation captions:

1. Some loop expressions use a relational operator like userNum > 0. If the input is 902, the first iteration executes (902 > 0), so outputs 902 % 10, or 2.
2. The body assigned userNum = userNum / 10, so 902 becomes 90. The loop jumps back, and since 90 > 0, the loop iterates again, outputting 90 % 10 or 0.
3. The body assigned userNum = userNum / 10, so 90 becomes 9. The loop jumps back, and since 9 > 0, the loop iterates again, outputting 9 % 10 or 9.
4. The body assigned userNum = userNum / 10, so 9 becomes 0. The loop jumps back, and since 0 > 0 is false, execution proceeds past the loop.

PARTICIPATION ACTIVITY 4.3.6: Loop expressions.

Use a *single operator* in each loop expression, and the most straightforward translation of the stated goal into an expression.

```
while ( _____ ) {
    // Loop body
}
```

1) Iterate while x is less than 100.

Check Show answer

2) Iterate while x is greater than or equal to 0.

Check Show answer

3) Iterate while c equals 'g'.

Check Show answer

4) Iterate while c is not equal to 'x'.

Check Show answer

5) Iterate *until* c equals 'z' (tricky; think carefully).

Check Show answer

Example: Ancestors

Below is another loop example. The program asks the user to enter a year, and then outputs the approximate number of a person's ancestors who were alive for each generation leading back to that year, with the loop computing powers of 2 along the way.

Figure 4.3.3: While loop example: Ancestors printing program.

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

int main() {
   const int YEARS_PER_GEN = 20; // Approx. years per generation
   int userYear;                 // User input
   int consYear;                 // Year being considered
   int numAnc;                   // Approx. ancestors in considered year

   consYear = 2020;
   numAnc = 2;

   cout << "Enter a past year (neg. for B.C.): ";
   cin >> userYear;

   while (consYear >= userYear) {
      cout << "Ancestors in " << consYear << ": " << numAnc << endl;

      numAnc = 2 * numAnc;                 // Each ancestor had two parents
      consYear = consYear - YEARS_PER_GEN; // Go back 1 generation
   }

   return 0;
}
```

```
Enter a past year (neg. for B.C.): 1900
Ancestors in 2020: 2
Ancestors in 2000: 4
Ancestors in 1980: 8
Ancestors in 1960: 16
Ancestors in 1940: 32
Ancestors in 1920: 64
Ancestors in 1900: 128

...

Enter a past year (neg. for B.C.): 1600
Ancestors in 2020: 2
Ancestors in 2000: 4
Ancestors in 1980: 8
Ancestors in 1960: 16
Ancestors in 1940: 32
Ancestors in 1920: 64
Ancestors in 1900: 128
Ancestors in 1880: 256
Ancestors in 1860: 512
Ancestors in 1840: 1024
Ancestors in 1820: 2048
Ancestors in 1800: 4096
Ancestors in 1780: 8192
Ancestors in 1760: 16384
Ancestors in 1740: 32768
Ancestors in 1720: 65536
Ancestors in 1700: 131072
Ancestors in 1680: 262144
Ancestors in 1660: 524288
Ancestors in 1640: 1048576
Ancestors in 1620: 2097152
Ancestors in 1600: 4194304
```


©zyBooks 01/31/24 17:46 1939727
Rob Daglio
MDCCOP2335Spring2024

Each iteration prints a line with the year and the ancestors in that year. (Note: the numbers are large due to not considering breeding among distant relatives, but nevertheless a person has many ancestors).

PARTICIPATION ACTIVITY 4.3.7: Ancestors example.

Consider the example above.

1) The loop expression involves a relational operator.
   - True
   - False

2) The loop body updates the considered year consYear.
   - True
   - False

3) The user is asked to enter a value at the end of each loop iteration.
   - True
   - False

4) Each loop iteration outputs the current number of ancestors (numAnc), and then doubles numAnc in preparation for the next iteration.
   - True
   - False

Common errors

A common error is to use the opposite loop expression than desired, like using x == 0 rather than x != 0. Programmers should remember that the expression describes when the loop *should* iterate, not when the loop should terminate.

An ***infinite loop*** is a loop that never stops iterating. *A common error is to accidentally create an infinite loop, often by forgetting to update a variable in the body, or by creating a loop expression whose evaluation to false isn't always reachable.*

PARTICIPATION ACTIVITY 4.3.8: Infinite loops.

```
numKids = 2;

// Get userChar from input

while (userChar == 'y') {
  // Put numKids to output
  // numKids = numKids * 2;
}
```

Oops, forgot to get userChar from input again

```
// Get userVal from input

while (userVal != 0) {
  // Put userVal to output
  // userVal = userVal - 2;
}
```

6 4 2 0 *OK, loop terminates after 3 iterations*

3 1 -1 -3 -5 ... *Oops, loop expression always evaluates to true*

Animation content:

Two examples of infinite loops are shown side by side.
Static figure:
Begin code:
numKids = 2;

// Get userChar from input

while (userChar == 'y') {
 // Put numKids to output
 // numKids = numKids * 2;
}
End code.

Underneath is a quote from the programmer saying "Oops, forgot to get userChar from input again."

Static figure:
Begin code:
// Get userVal from input

while (userVal != 0) {
 // Put userVal to output
 // userVal = userVal - 2;
}
End code.

Underneath are two cases where the inputs differ. The first case uses input 6 with outputs 4, 2 and 0. The quote from the programmer says "OK, loop terminates after 3 iterations." The second case uses input 3 with outputs 1, -1, -3, -5, ... which is an infinite amount of outputs. The quote from the programmer says "Oops, loop expression always evaluates to true."

Animation captions:

1. This while loop gets userChar from input, iterating if userChar is 'y'. If the first input is 'y', the loop body executes a first time.
2. The loop body outputs numKids and updates numKids. But, the programmer forgot to get userChar from the input at the end of the loop body.
3. Thus, userChar is still 'y', and the loop body is executed again, and again, and again, with no way out. The loop is an "infinite loop".
4. Another cause is a bad loop expression. If userVal is 6, this loop iterates 3 times, for 6, 4, and 2.
5. ... But if userVal is 3, the loop iterates infinitely, for 3, 1, -1, -3, and so on.

For the above left, programmers must get in the habit of remembering to update needed variables at the loop body's end.

For the above right, *good practice is to include greater than or less than along with equality in a loop expression whenever possible*, such as userVal >= 0 rather than userVal != 0.

A program with an infinite loop may print excessively, or just seem to stall. On some systems, the user can halt execution by pressing Control-C on the command prompt, or by selecting Stop (or Pause) from within an IDE.

Another common error is to use the assignment operator = rather than the equality operator == in a loop expression, usually causing an unintended infinite loop.

PARTICIPATION ACTIVITY 4.3.9: While loop iterations.

What will the following code output? For an infinite loop, type "IL" (without the quotes).

1)
```
x = 0;

while (x > 0) {
   cout << x << " ";
   x = x - 1;
}
cout << "Bye";
```

Check Show answer

2)
```
x = 5;
y = 18;

while (y >= x) {
   cout << y << " ";
   y = y - x;
}
```

Check Show answer

3) (Assume the user always enters 'q').
```
z = 0;
c = 'y';

while (c = 'y') {
   cout << z << " ";
   cin >> c;
   z = z + 1;
}
```

Check Show answer

4)
```
x = 10;

while (x != 3) {
   cout << x << " ";
   x = x / 2;
}
```

Check Show answer

5)
```
x = 0;

while (x <= 5) {
   cout << x << " ";
}
```

Check Show answer

CHALLENGE ACTIVITY 4.3.1: Enter the output of the while loop.

539740.3879454.qx3zqy7

Start

Type the program's output

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

int main() {
   int g;

   g = 0;

   while (g <= 1) {
      cout << g << endl;
      g = g + 1;
   }

   return 0;
}
```

```
0
1
```


1 2 3 4 5

Check Next

CHALLENGE ACTIVITY 4.3.2: While loops.

539740.3879454.qx3zqy7

Start

A while loop reads floating-point numbers from input. Write an expression that executes the while loop until a floating-point number read from input is less than or equal to 10.0.

▶ **Click here for example**

Note: Your expression should describe when the loop should iterate, not when the loop should terminate.

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4
5  int main() {
6     double userIn;
7
8     cin >> userIn;
9
10    while (/* Your code goes here */) {
11       cout << fixed << setprecision(1) << "Input is " << userIn << endl;
12       cin >> userIn;
13    }
14
15    cout << "Done" << endl;
```


1 2 3

Check Next level

4.4 More while examples

Example: GCD

The following is an example of using a loop to compute a mathematical quantity. The program computes the greatest common divisor (GCD) among two user-entered integers numA and numB, using Euclid's algorithm: If numA > numB, set numA to numA - numB, else set numB to numB - numA. These steps are repeated until numA equals numB, at which point numA and numB each equal the GCD.

Figure 4.4.1: While loop example: GCD (greatest common divisor) program.

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

// Output GCD of user-input numA and numB

int main() {
   int numA;  // User input
   int numB;  // User input

   cout << "Enter first positive integer: ";
   cin  >> numA;

   cout << "Enter second positive integer:
";
   cin  >> numB;

   while (numA != numB) { // Euclid's
algorithm
      if (numB > numA) {
         numB = numB - numA;
      }
      else {
         numA = numA - numB;
      }
   }

   cout << "GCD is: " << numA << endl;

   return 0;
}
```

```
Enter first positive integer: 9
Enter second positive integer: 7
GCD is: 1

...

Enter first positive integer: 15
Enter second positive integer:
10
GCD is: 5

...

Enter first positive integer: 99
Enter second positive integer:
33
GCD is: 33

...

Enter first positive integer:
500
Enter second positive integer:
500
GCD is: 500
```


PARTICIPATION ACTIVITY 4.4.1: GCD program.

Refer to the GCD code above. Assume user input of numA = 15 and numB = 10.

1) For the GCD program, what is the value of numA *before* the first loop iteration?

Check Show answer

2) What is the value of numB *after* the first iteration of the while loop?

Check Show answer

3) What is numB after the second iteration of the while loop?

Check Show answer

4) How many loop iterations will the algorithm execute?

Check Show answer

Example: Conversation

Below is a program that has a "conversation" with the user, asking the user to type something and then (randomly) printing one of four possible responses until the user enters "Goodbye".

The program uses getline() to read a string from the user into userText. The loop checks whether userText is "Goodbye"; if not, the loop body executes. The loop body generates a "random" number between 0 and 3, by using .size() to get the length of the user's string, which can vary, and mod'ing the length by 4. The loop body then prints one of four messages, using an if-else statement.

Figure 4.4.2: While loop example: Conversation program.

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

/* Program that has a conversation with the user.
   Uses a if-else statements and a random number (sort of)
   to mix up the program's responses. */

int main() {
   int randNum0_3;          // Random number 0 to 3
   string userText;         // User input

   cout << "Tell me something about yourself." << endl;
   cout << "You can type \"Goodbye\" at anytime to quit."
        << endl << endl << "> ";

   getline(cin, userText);

   while (userText != "Goodbye") {
      randNum0_3 = userText.size() % 4; // "Random" num. %4 ensures 0-3

      if (randNum0_3 == 0) {
         cout << endl << "Please explain further."
              << endl << endl << "> ";
      }
      else if (randNum0_3 == 1) {
         cout << endl << "Why do you say: \"" << userText << "\"?"
              << endl << endl << "> ";
      }
      else if (randNum0_3 == 2) {
         cout << endl << "I don't think that's right."
              << endl << endl << "> ";
      }
      else if (randNum0_3 == 3) {
         cout << endl << "What else can you share?"
              << endl << endl << "> ";
      }
      else {
         cout << endl << "Uh-oh, something went wrong. Try again."
              << endl << endl;
      }

      getline(cin, userText);
   }

   cout << endl << "It was nice talking with you. Goodbye." << endl;

   return 0;
}
```

```
Tell me something about yourself.
You can type "Goodbye" at anytime to quit.

> I'm 26 years old.

Why do you say: "I'm 26 years old."?

> Well, I was born 26 years ago.

I don't think that's right.

> I am sure it is correct.

Please explain further.

> Goodbye

It was nice talking with you. Goodbye.
```

PARTICIPATION ACTIVITY 4.4.2: Conversation program.

1) What will be printed if the user types "Ouch"?

Check Show answer

2) What will be printed if the user types "Bye"?

Check Show answer

3) Which if-else branch will execute if the user types "Goodbye"? Valid answers are branch 0, 1, 2, 3, or none.

Check Show answer

4) How many loop iterations will execute if the user plans to type "I'm hungry", "You are weird", "Goodbye", and "I like you".

Check Show answer

Example: Getting input until a sentinel is seen

Loops are commonly used to process an input list of values. A ***sentinel value*** is a special value indicating the end of a list, such as a list of positive integers ending with 0, as in 10 1 6 3 0. The example below computes the average of an input list of positive integers, ending with 0. The 0 is not included in the average.

Figure 4.4.3: Computing average of a list with a sentinel.

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

// Outputs average of list of positive integers
// List ends with 0 (sentinel)
// Ex: 10 1 6 3 0  yields (10 + 1 + 6 + 3) / 4, or 5

int main() {
   int currValue;
   int valuesSum;
   int numValues;

   valuesSum = 0;
   numValues = 0;

   cin >> currValue;

   while (currValue > 0) { // Get values until 0 (or less)
      valuesSum = valuesSum + currValue;
      numValues = numValues + 1;
      cin >> currValue;
   }

   cout << "Average: " << (valuesSum / numValues) << endl;

   return 0;
}
```

```
10 1 6 3 0
Average: 5

...

90 70 30 10 99 1 0
Average: 50
```


PARTICIPATION ACTIVITY 4.4.3: Average example with a sentinel.

Consider the example above.

1) How many actual (non-sentinel) values are given in the first input sequence?
   - 1
   - 4
   - 5

2) For the first input sequence, what is the final value of numValues?
   - 0
   - 4
   - 5

3) Suppose the first input was 0. Would valuesSum / numValues be 0?
   - Yes
   - No

4) What would happen if this list was input: 10 1 6 3 -1

- Output would be 5
- Output would be 4
- Error

CHALLENGE ACTIVITY 4.4.1: While loop with sentinel.

539740.3879454.qx3zqy7

Start

Type the program's output

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

int main() {
   int userValue;
   int minNumber;

   cin >> userValue;

   minNumber = userValue;

   while (userValue > 0) {
      if (userValue < minNumber) {
         minNumber = userValue;
      }
      cin >> userValue;
   }

   cout << "Min value: " << minNumber;

   return 0;
}
```

Input

```
44 23 56 12 0
```

Output

```
Min value: 12
```

1 2 3

Check Next

CHALLENGE ACTIVITY 4.4.2: Bidding example.

Variable keepBidding is initially assigned with 'y'. Within the while loop, keepBidding is updated with the user's next input value. Complete the while loop's expression to terminate the while loop if the user enters 'n'.

Ex: If the input is `y y n`, then the output is:

```
I'll bid $3!
Continue bidding? (y/n) I'll bid $6!
Continue bidding? (y/n) I'll bid $9!
Continue bidding? (y/n)
```


Learn how our autograder works

539740.3879454.qx3zqy7

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

int main() {
   char keepBidding;
   int nextBid;

   nextBid = 0;
   keepBidding = 'y';

   while (/* Your solution goes here  */) {
      nextBid = nextBid + 3;
      cout << "I'll bid $" << nextBid << "!" << endl;
      cout << "Continue bidding? (y/n) ";
      cin >> keepBidding;
```


Run

CHALLENGE ACTIVITY 4.4.3: While loop: Insect growth.

Given positive integer numInsects, write a while loop that prints, then doubles, numInsects each iteration. Print values < 200. Follow each number with a space. After the loop, print a newline. Ex: If numInsects = 16, print:

```
16 32 64 128
```

Learn how our autograder works

539740.3879454.qx3zqy7

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

int main() {
   int numInsects;

   cin >> numInsects; // Must be >= 1

   /* Your solution goes here  */

   return 0;
}
```


Run

CHALLENGE ACTIVITY 4.4.4: While loops with numbers.

539740.3879454.qx3zqy7

Start

Use cin to read integers from input into inputValue until -99 is read. For each remaining integer read before -99, if the integer equal to -17, output "Value is detected" followed by a newline and increment numCounts.

Ex: If the input is `-17 -18 -17 -17 -99`, then the output is:

```
Value is detected
Value is detected
Value is detected
3 time(s)
```

Note: The sentinel value is -99.


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

int main() {
   int inputValue;
   int numCounts;

   numCounts = 0;

   /* Your code goes here */

   cout << numCounts << " time(s)" << endl;

   return 0;
}
```

1 2

Check Next level

CHALLENGE ACTIVITY 4.4.5: Advanced while loop examples.

539740.3879454.qx3zqy7

Start

Integer griddleCount is initialized with 0. String inputItem is read from input. Write a while loop that iterates until inputItem is to "Stop". In each iteration of the loop:

- Increment griddleCount if inputItem is equal to "Griddle".
- Read string inputItem from input.

▶ **Click here for example**


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

int main() {
   string inputItem;
   int griddleCount;

   griddleCount = 0;

   cin >> inputItem;

```

```
12    /* Your code goes here */
13
14    cout << "Griddle occurs " << griddleCount << " time(s)." << endl;
15
```

1 2 3

Check Next level

4.5 Do-while loops

A ***do-while loop*** is a loop construct that first executes the loop body's statements, then checks the loop condition.

Construct 4.5.1: Do-while loop.

```
do {
   // Loop body
} while
(loopExpression);
```

Versus a while loop, a do-while loop is useful when the loop should iterate at least once.

PARTICIPATION ACTIVITY 4.5.1: Do-while loop.

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

int main() {
  char fill;

  fill = '*';

  do {
     cout << fill << fill << fill << endl;
     cout << fill << fill << fill << endl;
     cout << fill << fill << fill << endl;

     cout << "Enter char (q to quit): ";
     cin >> fill;
     cout << endl;
  } while (fill != 'q');

  return 0;
}
```

```
***
***
***
Enter char (q to quit):  x

xxx
xxx
xxx

Enter char (q to quit): q
```


Animation content:

Static figure:
Begin C++ code.

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

int main() {
   char fill;

   fill = '*';

   do {
      cout << fill << fill << fill << endl;
      cout << fill << fill << fill << endl;
      cout << fill << fill << fill << endl;

      cout << "Enter char (q to quit): ";
      cin >> fill;
      cout << endl;
   } while (fill != 'q');

   return 0;
}
```

End C++ code.

The display box is used to display the output of the program and user-input. The display box is empty.

Step 1: The program executes the loop body's statements first, then checks the loop condition. The line of code, "fill = '*';", is highlighted and character fill is assigned with '*'. Then, the first three lines of the do-while loop are highlighted, each containing the line of code "cout << fill << fill << fill << endl;". The display box now contains three lines of output:

```
***
***
***
```

The line of code, "cout << "Enter char (q to quit): ";" is highlighted and "Enter char (q to quit): " is output to the fourth line of the display box. The lines of code, "cin >> fill;" and "cout << endl;", are highlighted and the user inputs 'x' to the display box. Character fill is assigned with 'x' and the display box now contains four lines of output:

```
***
***
***
Enter char (q to quit): x
```

Step 2: Because x != 'q' evaluates to true, a second iteration will occur. The line of code, " } while (fill != 'q');" is highlighted and returns true. Thus, the first three lines of the do-while loop are highlighted for a second time, each containing the line of code "cout << fill << fill << fill << endl;". The display box now contains seven lines of output:


```
***
***
***
Enter char (q to quit): x
xxx
xxx
```

xxx

The line of code, "cout << "Enter char (q to quit): ";" is highlighted and "Enter char (q to quit): " is output to the eighth line of the display box. The lines of code, "cin >> fill;" and "cout << endl;", are highlighted again, but this time the user inputs 'q' to the display box. Character fill is assigned with 'q' and the display box now contains eight lines of output:

```
***
***
***
Enter char (q to quit): x
xxx
xxx
xxx
Enter char (q to quit): q
```

Step 3: Because q != 'q' evaluates to false, execution proceeds past the loop. The line of code, " } while (fill != 'q');" is highlighted and returns false. Thus, the do-while loop terminates.

Animation captions:

1. The program executes the loop body's statements first, then checks the loop condition.
2. Because x != 'q' evaluates to true, a second iteration will occur.
3. Because q != 'q' evaluates to false, execution proceeds past the loop.

PARTICIPATION ACTIVITY 4.5.2: Do-while loop.

Consider the following loop:

```
count = 0;
num = 6;

do {
   num = num - 1;
   count = count + 1;
} while (num > 4);
```

1) What is the value of count after the loop?
   - 0
   - 1
   - 2

2) What initial value of num would prevent count from being incremented?
   - 4
   - 0
   - No such value.

CHALLENGE ACTIVITY 4.5.1: Basic do-while loop with user input.

Complete the do-while loop to output from 0 to the value of countLimit using printVal. Assume the user will only input a positive number.
For example, if countLimit is 5 the output will be

```
0 1 2 3 4 5
```

Learn how our autograder works

539740.3879454.qx3zqy7

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

int main() {
   int countLimit;
   int printVal;

   // Get user input
   cin >> countLimit;

   printVal = 0;
   do {
      cout << printVal << " ";
      printVal = printVal + 1;
   } while ( /* Your solution goes here  */ );
```

Run

CHALLENGE ACTIVITY 4.5.2: Do-while loop to prompt user input.

Write a do-while loop that continues to prompt a user to enter a number less than 100, until the entered number is actually less than 100. End each prompt with a newline. Ex: For the user input 123, 395, 25, the expected output is:

```
Enter a number (<100):
Enter a number (<100):
Enter a number (<100):
Your number < 100 is: 25
```

Learn how our autograder works

539740.3879454.qx3zqy7

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

int main() {
   int userInput;

   /* Your solution goes here  */

   cout << "Your number < 100 is: " << userInput << endl;

   return 0;
}
```

Run

4.6 For loops

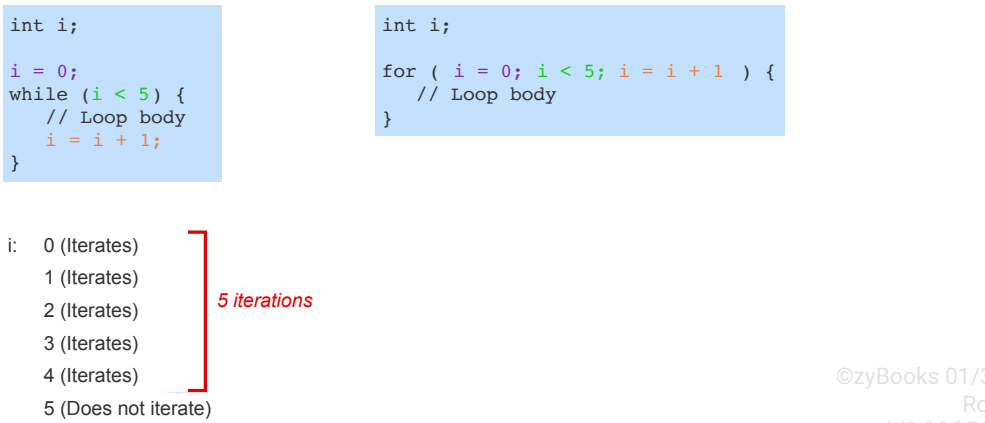
Basics

A loop commonly must iterate a specific number of times, such as 10 times. Though achievable with a while loop, that situation is so common that a special kind of loop exists. A ***for loop*** is a loop with three parts at the top: a loop variable initialization, a loop expression, and a loop variable update. A for loop describes iterating a specific number of times more naturally than a while loop.

Construct 4.6.1: For loop.

```
for (initialExpression; conditionExpression;
updateExpression) {
   // Loop body
}
// Statements after the loop
```

PARTICIPATION ACTIVITY 4.6.1: For loops.



Animation content:

Static figure:

Two code snippets are shown, one using a while loop and the other using a for loop.

Begin code:
int i;

i = 0;
while (i < 5) {
   // Loop body
   i = i + 1;
}
End code.

Begin code:
int i;

for (i = 0; i < 5; i = i + 1) {
   // Loop body
}
End code.

A walkthrough of the code for each value of the variable i is shown.

At i = 0, 1, 2, 3, and 4, the loop iterates (5 iterations). At i = 5, the loop does not iterate.

Animation captions:

1. This while loop pattern with i = 0 before, loop expression i < 5, and loop body ending with i = i + 1, iterates 5 times: when i = 0, 1, 2, 3, and 4.
2. The pattern is so common that a special construct, a for loop, exists to collect the three parts in one place at the loop's top, improving readability and reducing errors.
3. Note that semicolons separate the three parts. No semicolon is needed at the end.

The statement i = i + 1 is so common that the language supports the shorthand ***++i***, with ++ known as the ***increment operator***. (Likewise, ***--*** is the ***decrement operator***, --i means i = i - 1). As such, a standard way to loop N times is shown below.

Figure 4.6.1: A standard way to loop N times, using a for loop.

```
int i;
...
for (i = 0; i < N; ++i)
{
   ...
}
```


PARTICIPATION ACTIVITY 4.6.2: For loops.

1) What are the values of i for each iteration of:

```
for (i = 0; i < 6; ++i) {
   ...
}
```

- 1, 2, 3, 4, 5
- 0, 1, 2, 3, 4, 5
- 0, 1, 2, 3, 4, 5, 6

2) How many times will this loop iterate?

```
for (i = 0; i < 8; ++i) {
   ...
}
```

- 7 times
- 8 times
- 9 times

3) Goal: Loop 10 times

```
for (i = 0; _____; ++i) {
   ...
}
```

- i < 9
- i < 10
- i < 11

4) Goal: Loop 99 times

```
for (i = 0; _____; ++i) {
   ...
}
```

- i < 99
- i <= 99
- i == 99

5) Goal: Loop 20 times

```
for (____; i < 20; ++i) {
   ...
}
```

- i = 0
- i = 1

6) Goal: Loop numYears times (numYears is an int variable).

```
for (i = 0; _____; ++i) {
   ...
}
```


- numYears
- i <= numYears
- i < numYears

PARTICIPATION ACTIVITY 4.6.3: For loops.

Write for loops using the following form:

```
for (i = 0; i < 10; ++i) {
```

Note: Using a variable other than i is not accepted by this activity.

1) Complete the for loop to iterate 5 times. (Don't forget the semicolon).

```
for (i = 0;
                ++i) {
   ...
}
```

Check Show answer

2) Complete the for loop to iterate 7 times.

```
for (
++i) {
   ...
}
```

Check Show answer

3) Complete the for loop to iterate 500 times. (Don't forget the parentheses).

```
for                 {
   ...
}
```

Check Show answer

4) Complete the for loop to iterate numDogs times. numDogs is an int variable.

```
for (i = 0;
                ++i) {
   ...
}
```

Check Show answer

©zyBooks 01/31/24 17:46 1939727
Rob Daglio
MDCCOP2335Spring2024

*Note: Actually two increment operators exist: ++i (**pre-increment**) and i++ (**post-increment**). ++i increments before evaluating to a value, while i++ increments after. Ex: If i is 5, outputting ++i outputs 6, while outputting i++ outputs 5 (and then i becomes 6). This material primarily uses ++i for simplicity and safety, although many programmers use i++, especially in for loops.*

Example: Savings with interest

The following program outputs the amount of a savings account each year for 10 years, given an input initial amount and interest rate. A for loop iterates 10 times, such that each iteration represents one year, outputting that year's savings amount.

Figure 4.6.2: For loop: Savings interest program.

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

int main() {
   double initialSavings;  // User-entered initial
savings
   double interestRate;    // Interest rate
   double currSavings;     // Current savings with
interest
   int i;                  // Loop variable

   cout << "Enter initial savings: ";
   cin  >> initialSavings;

   cout << "Enter interest rate: ";
   cin  >> interestRate;

   cout << endl << "Annual savings for 10 years: " <<
endl;

   currSavings = initialSavings;
   for (i = 0; i < 10; ++i) {
      cout << "$" << currSavings << endl;
      currSavings = currSavings + (currSavings *
interestRate);
   }

   return 0;
}
```

```
Enter initial savings:
10000
Enter interest rate:
0.05

Annual savings for 10
years:
$10000
$10500
$11025
$11576.2
$12155.1
$12762.8
$13401
$14071
$14774.6
$15513.3
```

PARTICIPATION ACTIVITY 4.6.4: Savings interest program.

Consider the example above.

1) How many times does the for loop iterate?

   - 5
   - 10

2) During each iteration, the loop body's statements output the current savings amount, and then _____.

- increment i
- update currSavings

3) Can the input values change the number of loop iterations?

- Yes
- No

Example: Computing the average of a list of input values

The example below computes the average of an input list of integer values. The first input indicates the number of values in the subsequent list. That number controls how many times the subsequent for loop iterates.

Figure 4.6.3: Computing an average, with first value indicating list size.

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

// Outputs average of list of integers
// First value indicates list size
// Ex: 4  10 1 6 3  yields (10 + 1 + 6 + 3) / 4, or 5

int main() {
   int currValue;
   int valuesSum;
   int numValues;
   int i;

   cin >> numValues; // Gets number of values in list

   valuesSum = 0;

   for (i = 0; i < numValues; ++i) {
      cin >> currValue; // Gets next value in list
      valuesSum += currValue;
   }

   cout << "Average: " << (valuesSum / numValues) <<
endl;

   return 0;
}
```

```
4 10 1 6 3
Average: 5

...

5 -75 -50 30 60
80
Average: 9
```


PARTICIPATION ACTIVITY 4.6.5: Computing the average.

Consider the example above, with input 4 10 1 6 3. Note: The first input indicates the number of values in the subsequent list.

1) Before the loop is entered, what is valuesSum?

Check Show answer

2) What is valuesSum after the first iteration?

Check Show answer

3) What is valuesSum after the second iteration?

Check Show answer

4) valuesSum is 20 after the fourth iteration. What is numValues?

Check Show answer

5) For the following input, how many times will the for loop iterate?
7 -1 -3 -5 -14 -15 -20 -40

Check Show answer

Choosing among for and while loops

Generally, a programmer uses a for loop when the number of iterations is known (like loop 5 times, or loop numItems times), and a while loop otherwise.

Table 4.6.1: Choosing between while and for loops: General guidelines (not strict rules though).

for	Number of iterations is computable before the loop, like iterating N times.
while	Number of iterations is not (easily) computable before the loop, like iterating until the input is 'q'.

PARTICIPATION ACTIVITY 4.6.6: While loops and for loops.

Choose the most appropriate loop type.

1) Iterate as long as user-entered char c is not 'q'.
   - while
   - for

2) Iterate until the values of x and y are equal, where x and y are changed in the loop body.
   - while
   - for

3) Iterate 100 times.
   - while
   - for

CHALLENGE ACTIVITY 4.6.1: Enter the for loop's output.

539740.3879454.qx3zqy7

Start

Type the program's output

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

int main() {
   int i;

   for (i = 0; i < 2; ++i) {
      cout << i;
   }

   return 0;
}
```

01

1 2 3

Check Next

CHALLENGE ACTIVITY 4.6.2: Read and format integers.

539740.3879454.qx3zqy7

Start

Integer valCount is read from input representing the number of integers to be read next. Use a loop to read the remaining int from input into variable value. For each integer from 0 to valCount minus 1, inclusive, output "Read is " followed by the intege each output with a newline.

▶ **Click here for example**

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

int main() {
   int valCount;
   int value;
   int i;

   cin >> valCount;

   /* Your code goes here */

   return 0;
}
```


1 2 3

Check Next level

4.7 More for loop examples

Example: Finding the max

Analyzing data is a common programming task. A common data analysis task is to find the maximum value in a list of values. A loop can achieve that task by updating a max-seen-so-far variable on each iteration.

Figure 4.7.1: Finding the max in a list.

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

// Outputs max of list of integers
// First value indicates list size
// Ex: 4 -1 9 0 3  yields 9

int main() {
   int maxSoFar;
   int currValue;
   int numValues;
   int i;

   cin >> numValues;

   for (i = 0; i < numValues; ++i) {
      cin >> currValue;

      if (i == 0) { // First iteration
         maxSoFar = currValue;
      }
      else if (currValue > maxSoFar) {
         maxSoFar = currValue;
      }
   }

   if (numValues > 0) {
      cout << "Max: " << maxSoFar <<
endl;
   }

   return 0;
}
```

```
4 -1 9 0 3
Max: 9

...

5 -15 -90 -2 -60
-30
Max: -2
```


PARTICIPATION ACTIVITY 4.7.1: Finding the max.

Consider the example above.

1) Before entering the loop, what is the maximum value seen so far from the list of integers?
   - 0
   - -1
   - No such value

2) The loop's first iteration gets the list's first integer into variable currValue. Is that the maximum value seen so far?
   - Yes
   - No

3) For each iteration after the first iteration, the comparison _____ is checked.

- maxSoFar > currValue
- currValue > maxSoFar
- currValue == maxSoFar

Beyond iterating N times

The three parts of a for loop may be adjusted to do more than just iterate N times. For example, a for loop can output various sequences. The following outputs multiples of 5 from 10 to 50.

Figure 4.7.2: Outputting multiples of 5 from 10 to 50.

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

// Outputs 10 15 20 25 30 35 40 45
50

int main() {
   int i;

   for (i = 10; i <= 50; i = i + 5)
{
      cout << i << " ";
   }

   cout << endl;

   return 0;
}
```

```
10 15 20 25 30 35 40 45
50
```

PARTICIPATION ACTIVITY 4.7.2: For loops beyond iterating N times.

Type the output of the for loop. Whitespace matters, including after the last item.

Example:

```
for (i = 1; i <= 5; ++i) {
  (Put i to output, followed by a space)
}
```

Outputs:
1 2 3 4 5

1)
```
for (i = 0; i < 5; ++i) {
  (Put i to output, followed
by a space)
}
```

Check Show answer

2)
```
for (i = 1; i <= 5; ++i) {
  (Put i to output, followed
by a space)
}
```

Check Show answer

3)
```
for (i = 0; i < 10; i = i +
2) {
  (Put i to output, followed
by a space)
}
```

Check Show answer

4)
```
for (i = -3; i <= 3; ++i) {
  (Put i to output, followed
by a space)
}
```

Check Show answer

5)
```
for (i = 5; i >= 0; --i) {
  (Put i to output, followed
by a space)
}
```

Check Show answer

6)
```
for (i = 0; i < 5; ++i) {
  (Put 2 * i to output,
followed by a space)
}
```

Check Show answer

Example: Outputting a table of temperatures

Programs are sometimes used to auto-generate data tables. The following program generates a table of Celsius and Fahrenheit temperature values, in increments of 5 C. The for loop counts from -10 to 40 in increments of 5, and names the loop variable currC rather than i to be more descriptive.

Figure 4.7.3: Auto-generate a data table: Celsius to Fahrenheit.

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

int main() {
   int currC;
   double equivalentF;

   for (currC = -10; currC <= 40; currC += 5)
{
      equivalentF =  (currC * 9.0 / 5.0) +
32.0;

      cout << currC << " C is ";
      cout << equivalentF << " F";
      cout << endl;
   }

   return 0;
}
```

```
-10 C is 14
F
-5 C is 23 F
0 C is 32 F
5 C is 41 F
10 C is 50 F
15 C is 59 F
20 C is 68 F
25 C is 77 F
30 C is 86 F
35 C is 95 F
40 C is 104
F
```


PARTICIPATION ACTIVITY 4.7.3: For loop generating a table of temperature values.

Consider the example above.

1) What is the loop variable's name?

Check Show answer

2) What are the values of currC for the first four iterations?
Type as: 1 9 2 6

Check Show answer

3) What is the loop expression? (The expression checked for whether to enter the loop body).

Check Show answer

Loop style issues

Starting with 0

Programmers in C, C++, Java, and other languages have generally standardized on looping N times by starting with i = 0 and checking for i < N, rather than by using i = 1 and i <= N. One reason is due to other constructs (arrays / vectors), often used with loops, start with 0. Another is simply that a choice was made.

The ++ operators

The ++ operator can appear as ++i (***prefix form***) or as i++ (***postfix form***). ++i increments i first and then evaluates the result, while i++ evaluates the result first and then increments i. The distinction is relevant in a statement like x = ++i vs. x = i++; if i is 5, the first yields x = 6, the second x = 5.

Some consider ++i safer for beginners in case they type i = ++i, which typically works as expected (whereas i = i++ does not), so this material uses ++i throughout. The -- operator also has prefix and postfix versions. Incidentally, the C++ programming language gets its name from the ++ operator, suggesting C++ is an increment or improvement over its C language predecessor.

In-loop declaration of i

Variables can be declared throughout code, so many programmers use: `for (int i = 0; i < N; ++i)`. But, the teaching experience of this material's authors suggests such declarations may confuse learners who may declare variables within loops, repeatedly re-declaring variables, etc. This material avoids the in-loop declaration approach. The authors hope to make the learning less error-prone, and have confidence that programmers can easily pick up on the common in-loop declaration approach later.

PARTICIPATION ACTIVITY 4.7.4: Miscellaneous for loop and ++ topics.

1) Do these loops iterate the same number of times?

```
for (i = 0; i < 5; ++i) {
   ...
}

for (i = 1; i <= 5; ++i) {
   ...
}
```

- Yes
- No

2) Does this for loop iterate 5 times?

```
for (i = 0; i < 5; i++) {
   ...
}
```

- Yes
- No

3) Is the following valid code?

```
for (int i = 0; i < 5; ++i) {
   ...
}
```

- Yes
- No

Common errors / good practice

A common error is to also have a ++i; statement in the loop body, causing the loop variable to be updated twice per iteration.

Figure 4.7.4: Common error: loop variable updated twice.

```
// Loop variable updated twice per
iteration
for (i = 0; i < 5; ++i) {
   // Loop body
   ++i; // Oops
}
```


While the initialization and update parts of a for loop can include multiple statements separated by a comma, good practice is to use a single statement for each part. Good practice also is to use a for loop's parts to count the necessary loop iterations, with nothing added or omitted. The following loop examples should be avoided, if possible.

Figure 4.7.5: Avoid these for loop variations.

```
// initialExpression not related to counting iterations; move r = rand()
before loop
for (i = 0, r = rand(); i < 5; ++i) {
   // Loop body
}

// updateExpression not related to counting iterations; move r = r + 2 into
loop body
for (i = 0; i < 5; ++i, r = r + 2) {
   // Loop body
}
```

PARTICIPATION ACTIVITY 4.7.5: For loop: Common errors / good practice.

1) Putting ++i at the end of a for loop body, in addition to in the updateExpression part, yields a syntax error.
   - ○ True
   - ○ False

2) The above two for loop variations each yield a syntax error.
   - ○ True
   - ○ False

CHALLENGE ACTIVITY 4.7.1: For loops.

539740.3879454.qx3zqy7

Start

Write a for loop that prints: 1 2 ... numVal

Ex: If the input is:

```
4
```

the output is:

```
1 2 3 4
```

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

int main() {
   int numVal;
   int i;

   cin >> numVal;

   for(/* Your code goes here */) {
      cout << i << " ";
   }

   return 0;
}
```


1 2 3 4 5

Check Next

CHALLENGE ACTIVITY 4.7.2: Find extreme value.

539740.3879454.qx3zqy7

Start

Assign variable lowestVal with the smallest value of 9 positive floating-point values read from input.

Ex: If the input is:

```
1.8 8.1 2.6 2.4 15.5 14.1 14.6 1.3 16.2
```

then the output is:

```
1.3
```

Note:

- The first value read is the smallest value seen so far.
- All floating-point values are of type double.

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

int main() {
   double inValue;
   double lowestVal;
   int i;

   /* Your code goes here */

   cout << lowestVal << endl;
```

```
13
14    return 0;
15 }
```

1 2

Check Next level

CHALLENGE ACTIVITY 4.7.3: All values fit a category?

539740.3879454.qx3zqy7

Start

Declare a Boolean variable named allOdd. Then, read integer valCount from input representing the number of integers to be r next. Use a loop to read the remaining integers from input. If all valCount integers are odd, assign allOdd with true. Otherwise assign allOdd with false.

Code at the end of main() outputs "All match" if allOdd is true or "Not all match" if allOdd is false.

Ex: If the input is:

```
3
65 95 25
```

then the output is:

```
All match
```

Note: Odd integers are *not* divisible by 2.

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     int valCount;
6     /* Additional variable declarations go here */
7
8     /* Your code goes here */
9
10    if (allOdd) {
11       cout << "All match" << endl;
12    }
13    else {
14       cout << "Not all match" << endl;
15    }
```


1 2

Check Next level

4.8 Loops and strings

Iterating through a string with a for loop

A programmer commonly iterates through a string, examining each character. The following example counts the number of letters in a string, not counting digits, symbols, etc.

Figure 4.8.1: Iterating through a string: Counting letters.

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

int main() {
   string inputWord;
   int numLetters;
   unsigned int i;

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

   numLetters = 0;
   for (i = 0; i < inputWord.size(); ++i) {
      if (isalpha(inputWord.at(i))) {
         numLetters += 1;
      }
   }

   cout << "Number of letters: " << numLetters <<
endl;

   return 0;
}
```

```
Enter a word: Hey!!
Number of letters: 3

...

Enter a word:
123abc...xyz
Number of letters: 6
```

PARTICIPATION ACTIVITY 4.8.1: Iterating through a string.

1) To visit every character in a string, a for loop should iterate over indices ____.
   - 0 to size
   - 0 to size-1
   - 1 to size

2) If a for loop iterates through a string s using variable i, the loop body can access the current character as: s.at(____)
   - i-1
   - i
   - i+1

Iterating until done with a while loop

A programmer commonly wishes to iterate through a string until something is done. The example below replaces all occurrences of "U.S.A." with "USA". Because the number of iterations is not known beforehand, a while loop is used. The string functions find() and replace() are used to identify each instance of the "U.S.A." and replace each instance with "USA", respectively. Both functions are described in detail elsewhere.

Figure 4.8.2: Iterating until done: Replacing all occurrences of a word.


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

int main() {
   string userText;
   int usaIndex;

   cout << "Enter text: ";
   getline(cin, userText);

   // At least one occurrence exists
   while (userText.find("U.S.A.") !=
string::npos) {
      // Get index of first instance
      usaIndex = userText.find("U.S.A.");

      // U.S.A. is 6 long
      userText.replace(usaIndex, 6,
"USA");
   }

   cout << "New text: " << userText<<
endl;

   return 0;
}
```

```
Enter text: The U.S.A. is big. Are you
from the U.S.A.?
New text: The USA is big. Are you from
the USA?

...

Enter text: USA U.S.A. U.S.A.U.S.A.
Bye
New text: USA USA USAUSA Bye
```

PARTICIPATION ACTIVITY 4.8.2: Replacing until done.

Consider the example above.

1) The loop is entered as long as an occurrence of "U.S.A." ____ .
   - is found
   - is not found

2) The number of iterations is known before entering the loop.
   - True
   - False

3) find() is called within the loop body.
   - True
   - False

4) replace() is called with arguments being the index of the first occurrence of "U.S.A.", ____ , and "USA".

- -1
- 6

CHALLENGE ACTIVITY 4.8.1: Password requirements.

539740.3879454.qx3zqy7

Start

A website requires that passwords only contain letters. For each character in keyWord that is *not* a letter, replace the charact with 'x'.

Ex: If the input is `pf PK`, then the output is:

```
New password: pfxPK
```

Note: isalpha() returns true if a character is a letter, and false otherwise. Ex: isalpha('a') returns true. isalpha('8') returns false.

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

int main() {
   string keyWord;
   unsigned int i;

   getline(cin, keyWord);

   /* Your code goes here */

   cout << "New password: " << keyWord << endl;

   return 0;
```

1 2 3

Check Next level

4.9 Nested loops

A ***nested loop*** is a loop that appears in the body of another loop. The nested loops are commonly referred to as the ***inner loop*** and ***outer loop***.

Nested loops have various uses. One use is to generate all combinations of some items. For example, the following program generates all two-letter .com Internet domain names.

Figure 4.9.1: Nested loops example: Two-letter domain name printing program.

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

/* Output all two-letter .com Internet domain names
*/

int main() {
   char letter1;
   char letter2;

   cout << "Two-letter domain names:" << endl;

   letter1 = 'a';
   while (letter1 <= 'z') {
      letter2 = 'a';
      while (letter2 <= 'z') {
         cout << letter1 << letter2 << ".com" <<
endl;
         ++letter2;
      }
      ++letter1;
   }

   return 0;
}
```

```
Two-letter domain
names:
aa.com
ab.com
ac.com
ad.com
ae.com
af.com
ag.com
ah.com
ai.com
aj.com
ak.com
al.com
am.com
an.com
ao.com
ap.com
aq.com
ar.com
as.com
at.com
au.com
av.com
aw.com
ax.com
ay.com
az.com
ba.com
bb.com
bc.com
bd.com
be.com

...

zw.com
zx.com
zy.com
zz.com
```


Note that the program makes use of ascending characters being encoded as ascending numbers, e.g., 'a' is 97, 'b' is 98, etc., so assigning 'a' to letter1 and then incrementing yields 'b'.

(Forget about buying a two-letter domain name: They are all taken, and each sells for several hundred thousand or millions of dollars. Source: dnjournal.com, 2012).

zyDE 4.9.1: Two character dotcom domain names.

Modify the program to include two-character .com names where the second character can be a letter or a number, as in a2.com. Hint: Add a second loop, following the `while (letter2 <= 'z')` loop, to handle numbers.

Load default template... Run

```
1  #include <iostream>
2  using namespace std;
3
4  /* Output all two-letter
5
6  int main() {
7     char letter1;
8     char letter2;
9
10    cout << "Two-letter c
11
12    letter1 = 'a';
13    while (letter1 <= 'z'
14       letter2 = 'a';
15       while (letter2 <
```


Below is a nested loop example that graphically depicts an integer's magnitude by using asterisks, creating a "histogram." The inner loop is a for loop that handles the printing of the asterisks. The outer loop is a while loop that handles executing until a negative number is entered.

Figure 4.9.2: Nested loop example: Histogram.

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

int main() {
   int numAsterisk;  // Number of
asterisks to print
   int i;            // Loop counter

   numAsterisk = 0;

   while (numAsterisk >= 0) {
      cout << "Enter an integer (negative
to quit): ";
      cin >> numAsterisk;

      if (numAsterisk >= 0) {
         cout << "Depicted graphically:"
<< endl;
         for (i = 1; i <= numAsterisk;
++i) {
            cout << "*";
         }
         cout << endl << endl;
      }
   }

   cout << "Goodbye." << endl;

   return 0;
}
```

```
Enter an integer (negative to quit):
9
Depicted graphically:
*********

Enter an integer (negative to quit):
23
Depicted graphically:
***********************

Enter an integer (negative to quit):
35
Depicted graphically:
***********************************

Enter an integer (negative to quit):
-1
Goodbye.
```


PARTICIPATION ACTIVITY 4.9.1: Nested loops: Inner loop execution.

1) Given the following code, how many times will the inner loop body execute?

```
int row;
int col;

for(row = 0; row < 2; row =
row + 1) {
   for(col = 0; col < 3; col
= col + 1) {
      // Inner loop body
   }
}
```

Check Show answer

2) Given the following code, how many times will the inner loop body execute?

```
char letter1;
char letter2;

letter1 = 'a';
while (letter1 <= 'f') {
   letter2 = 'c';
   while (letter2 <= 'f') {
      // Inner loop body
      ++letter2;
   }
   ++letter1;
}
```

Check Show answer

PARTICIPATION ACTIVITY 4.9.2: Nested loops: What is the output.

1) What is output by the following code?

```
int row;
int col;

for(row = 2; row <= 3; row =
row + 1) {
   for(col = 0; col <= 1; col
= col + 1) {
      cout << row << col << "
";
   }
}
```

Check Show answer

2) What is output by the following code?

```
char letter1;
char letter2;

letter1 = 'y';
while (letter1 <= 'z') {
   letter2 = 'a';
   while (letter2 <= 'c') {
      cout << letter1 <<
letter2 << " ";
      ++letter2;
   }
   ++letter1;
}
```

Check Show answer

CHALLENGE ACTIVITY 4.9.1: Nested loops.

539740.3879454.qx3zqy7

Start

Integers outerRange and innerRange are read from input. The inner while loop executes innerRange times for each iteration outer while loop. Complete the outer while loop to execute (outerRange + 1) times.

Ex: If the input is `3 5`, then the output is:

```
Inner loop ran 20 times
```

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     int outerRange;
6     int innerRange;
7     int count;
8     int i;
9     int j;
10
11    cin >> outerRange;
12    cin >> innerRange;
13
14    count = 0;
15    i = 0;
```

1 2 3 4

Check Next level

4.10 Developing programs incrementally

Creating correct programs can be hard. Following a good programming process helps. What many new programmers do, but shouldn't, is write the entire program, compile it, and run it—hoping it works. Debugging such a program can be difficult because there may be many distinct bugs.

Experienced programmers develop programs ***incrementally***, meaning they create a simple program version, and then grow the program little-by-little into successively more-complete versions.

The following program allows the user to enter a phone number that includes letters. Such letters appear on phone keypads along with numbers, enabling phone numbers like 1-555-HOLIDAY. The program converts a phone number having numbers/letters into one having numbers only.

The first program version simply prints each string element, to ensure the loop iterates properly.

Figure 4.10.1: Incremental program development.

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

int main() {
   string phoneStr;    // User input: Phone number
string
   unsigned int i;     // Current element in phone
number string

   cout << "Enter phone number: ";
   cin >> phoneStr;

   for (i = 0; i < phoneStr.size(); ++i) { // For
each element
      cout << "Element " << i << " is: " <<
phoneStr.at(i) << endl;
   }

   return 0;
}
```

```
Enter phone number: 1-
555-HOLIDAY
Element 0 is: 1
Element 1 is: -
Element 2 is: 5
Element 3 is: 5
Element 4 is: 5
Element 5 is: -
Element 6 is: H
Element 7 is: O
Element 8 is: L
Element 9 is: I
Element 10 is: D
Element 11 is: A
Element 12 is: Y
```

The second program version outputs any number elements, outputting '?' for non-number elements. A ***FIXME comment*** is commonly used to indicate program parts to be fixed or added, as below. Some editor tools automatically highlight the FIXME comment to attract the programmer's attention.

Figure 4.10.2: Second version echoes numbers, and has FIXME comment.

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

int main() {
   string phoneStr;    // User input: Phone number
string
   unsigned int i;     // Current element in phone
number string
   char currChar;      // Current char in phone
number string

   cout << "Enter phone number: ";
   cin >> phoneStr;

   cout << "Numbers only: ";
   for (i = 0; i < phoneStr.size(); ++i) { // For
each element
      currChar = phoneStr.at(i);
      if ((currChar >= '0') && (currChar <= '9')) {
         cout << currChar; // Print element as is
      }
      // FIXME: Add else-if branches for letters
and hyphen
      else {
         cout << '?';
      }
   }

   cout << endl;

   return 0;
}
```

```
Enter phone number: 1-555-
HOLIDAY
Numbers only: 1?555????????
```


The third version completes the else-if branch for the letters A-C (lowercase and uppercase), per a standard phone keypad. The program also modifies the if branch to echo a hyphen in addition to numbers.

Figure 4.10.3: Third version echoes hyphens too, and handles first three letters.

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

int main() {
   string phoneStr;   // User input: Phone number
string
   unsigned int i;    // Current element in phone
number string
   char currChar;     // Current char in phone number
string

   cout << "Enter phone number: ";
   cin >> phoneStr;

   cout << "Numbers only: ";
   for (i = 0; i < phoneStr.size(); ++i) { // For
each element
      currChar = phoneStr.at(i);
      if (((currChar >= '0') && (currChar <= '9')) ||
(currChar == '-')) {
         cout << currChar; // Print element as is
      }
      else if ( ((currChar >= 'a') && (currChar <=
'c')) ||
                ((currChar >= 'A') && (currChar <=
'C')) ) {
         cout << "2";
      }
      // FIXME: Add remaining else-if branches
      else {
         cout << '?';
      }
   }

   cout << endl;

   return 0;
}
```

```
Enter phone number: 1-
555-HOLIDAY
Numbers only: 1-
555-??????2
```


The fourth version can be created by filling in the if-else branches similarly for other letters. We added more instructions too. Code is not shown below, but sample input/output is provided.

Figure 4.10.4: Fourth and final version sample input/output.

```
Enter phone number (letters/- OK, no spaces): 1-555-HOLIDAY
Numbers only: 1-555-4654329

...

Enter phone number (letters/- OK, no spaces): 1-555-holiday
Numbers only: 1-555-4654329

...

Enter phone number (letters/- OK, no spaces): 999-9999
Numbers only: 999-9999

...

Enter phone number (letters/- OK, no spaces): 9876zywx%$#@
Numbers only: 98769999????
```

zyDE 4.10.1: Incremental programming.

Complete the program by providing the additional if-else branches for decoding other letters in a phone number. Try incrementally writing the program by adding one "else if" branch at a time, testing that each added branch works as intended.

Load default template...


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

int main() {
   string phoneStr;   // User input: Phone number str
   unsigned int i;    // Loop index, current element
   char currChar;     // Current char in phone number

   cout << "Enter phone number: " << endl;
   cin >> phoneStr;

   cout << "Numbers only: ";
   for (i = 0; i < phoneStr.size(); ++i) { // For eac
      currChar = phoneStr.at(i);
```

1-800-555-HOLIDAY

Run

PARTICIPATION ACTIVITY 4.10.1: Incremental programming.

1) A good programming process is to write the entire program, then incrementally remove bugs one at a time.
   - True
   - False

2) Expert programmers need not develop programs incrementally.
   - True
   - False

3) Incremental programming may help reduce the number of errors in a program.

- True
- False

4) FIXME comments provide a way for a programmer to remember what needs to be added.

- True
- False

5) Once a program is complete, one would expect to see several FIXME comments.

- True
- False

4.11 Break and continue

A ***break statement*** in a loop causes an immediate exit of the loop. A break statement can sometimes yield a loop that is easier to understand.

Figure 4.11.1: Break statement: Meal finder program.

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

int main() {
   const int EMPANADA_COST = 3;
   const int TACO_COST     = 4;

   int userMoney;
   int numTacos;
   int numEmpanadas;
   int mealCost;
   int maxEmpanadas;
   int maxTacos;

   mealCost = 0;

   cout << "Enter money for meal: ";
   cin >> userMoney;

   maxEmpanadas = userMoney / EMPANADA_COST;
   maxTacos = userMoney / TACO_COST;

   for (numTacos = 0; numTacos <= maxTacos; ++numTacos) {
      for (numEmpanadas = 0; numEmpanadas <= maxEmpanadas; ++numEmpanadas) {

         mealCost = (numEmpanadas * EMPANADA_COST) + (numTacos * TACO_COST);

         // Find first meal option that exactly matches user money
         if (mealCost == userMoney) {
            break;
         }
      }

      // If meal option exactly matching user money is found,
      // break from outer loop as well
      if (mealCost == userMoney) {
         break;
      }
   }

   if (mealCost == userMoney) {
      cout << "$" << mealCost << " buys " << numEmpanadas
      << " empanadas and " << numTacos << " tacos without change." << endl;
   }
   else {
      cout << "You cannot buy a meal without having change left over." <<
endl;
   }

   return 0;
}
```

```
Enter money for meal: 20
$20 buys 4 empanadas and 2 tacos without change.

...

Enter money for meal: 31
$31 buys 9 empanadas and 1 tacos without change.
```


©zyBooks 01/31/24 17:46 1939727
Rob Daglio
MDCCOP2335Spring2024

The nested for loops generate all possible meal options for the number of empanadas and tacos that can be purchased. The inner loop body calculates the cost of the current meal option. If equal to the user's money, the search is over, so the break statement immediately exits the inner loop. The outer loop body also checks if equal, and if so that break statement exits the outer loop.

The program could be written without break statements, but the loop's condition expressions would be more complex and the program would require additional code, perhaps being harder to understand.

PARTICIPATION ACTIVITY 4.11.1: Break statements.

Given the following while loop, what is the value assigned to variable z for the given values of variables a, b and c?

```
mult = 0;

while (a < 10) {
   mult = b * a;
   if (mult > c) {
      break;
   }
   a = a + 1;
}
z = a;
```


1) a = 1, b = 1, c = 0

Check Show answer

2) a = 4, b = 5, c = 20

Check Show answer

A ***continue statement*** in a loop causes an immediate jump to the loop condition check. A continue statement can sometimes improve the readability of a loop. The example below extends the previous meal finder program to find meal options for which the total number of items purchased is evenly divisible by the number of diners. The program also outputs all possible meal options, instead of just reporting the first meal option found.

Figure 4.11.2: Continue statement: Meal finder program that ensures items purchased is evenly divisible by the number of diners.

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

int main() {
   const int EMPANADA_COST = 3;
   const int TACO_COST     = 4;

   int userMoney;
   int numTacos;
   int numEmpanadas;
   int mealCost;
   int maxEmpanadas;
   int maxTacos;
   int numOptions;
   int numDiners;

   mealCost = 0;
   numOptions = 0;

   cout << "Enter money for meal: ";
   cin >> userMoney;

   cout << "How many people are eating: ";
   cin >> numDiners;

   maxEmpanadas = userMoney / EMPANADA_COST;
   maxTacos     = userMoney / TACO_COST;

   for (numTacos = 0; numTacos <= maxTacos; ++numTacos) {
      for (numEmpanadas = 0; numEmpanadas <= maxEmpanadas; ++numEmpanadas) {

         // Total items purchased must be equally
         // divisible by number of diners
         if ( ((numTacos + numEmpanadas) % numDiners) != 0) {
            continue;
         }

         mealCost = (numEmpanadas * EMPANADA_COST) + (numTacos * TACO_COST);

         if (mealCost == userMoney) {
            cout << "$" << mealCost << " buys " << numEmpanadas
                 << " empanadas and " << numTacos
                 << " tacos without change." << endl;
            numOptions = numOptions + 1;
         }
      }
   }

   if (numOptions == 0) {
      cout << "You cannot buy a meal without "
           << "having change left over." << endl;
   }

   return 0;
}
```

```
Enter money for meal: 60
How many people are eating: 3
$60 buys 12 empanadas and 6 tacos without change.
$60 buys 0 empanadas and 15 tacos without change.

...

Enter money for meal: 54
How many people are eating: 2
$54 buys 18 empanadas and 0 tacos without change.
$54 buys 10 empanadas and 6 tacos without change.
$54 buys 2 empanadas and 12 tacos without change.
```

The nested loops generate all possible combinations of tacos and empanadas. If the total number of tacos and empanadas is not exactly divisible by the number of diners (e.g., `((numTacos + numEmpanadas) % numDiners) != 0)`, the continue statement proceeds to the next iteration, thus causing incrementing of numEmpanadas and checking of the loop condition.

Break and continue statements can avoid excessive indenting/nesting within a loop. But they could be easily overlooked, and should be used sparingly, when their use is clear to the reader.

PARTICIPATION ACTIVITY 4.11.2: Continue.

Given:

```
for (i = 0; i < 5; ++i) {
   if (i < 10) {
      continue;
   }
   // Put i to output
}
```

1) The loop will print at least some output.

- True
- False

2) The loop will iterate only once.

- True
- False

CHALLENGE ACTIVITY 4.11.1: Enter the output of break and continue.

539740.3879454.qx3zqy7

Start

Type the program's output

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

int main() {
   int stop;
   int n;

   cin >> stop;

   for (n = 0; n < 10; ++n) {
      if (n > stop) {
         cout << "n=" << n;
         cout << endl;
         break;
      }

      cout << n << endl;
   }

   return 0;
}
```

Input

7

Output

```
0
1
2
3
4
5
6
7
n=8
```

1 2 3 4 5

Check Next

CHALLENGE ACTIVITY 4.11.2: Simon says.

"Simon Says" is a memory game where "Simon" outputs a sequence of 10 characters (R, G, B, Y) and the user must repeat the sequence. Create a for loop that compares the two strings starting from index 0. For each match, add one point to userScore. Upon a mismatch, exit the loop using a break statement. Assume simonPattern and userPattern are always the same length. Ex: The following patterns yield a userScore of 4:

```
simonPattern: RRGBRYYBGY
userPattern:  RRGBBRYBGY
```

Learn how our autograder works

539740.3879454.qx3zqy7

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

int main() {
   string simonPattern;
   string userPattern;
   int userScore;
   int i;

   userScore = 0;

   cin >> simonPattern;
   cin >> userPattern;

```

Run

4.12 Variable name scope

Scope of names

A declared name is only valid within a region of code known as the name's ***scope***. Ex: A variable userNum declared in main() is only valid within main(), from the declaration to main()'s end.

Most of this material declares variables at the top of main() (and if the reader has studied functions, at the top of other functions). However, a variable may be declared within other blocks too. A ***block*** is a brace-enclosed {...} sequence of statements, such as found with an if-else, for loop, or while loop. A variable name's scope extends from the declaration to the closing brace }.

PARTICIPATION ACTIVITY 4.12.1: Variable name scope extends to the end of the declaration's block.

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

int main() {

   // int val1  = userNum;    // ERROR
  int userNum = 2;           // Name valid to main's "}"
  int newNum  = userNum + 1;
  int i;

   for (i = 0; i < newNum; ++i) {
     int valSquared;         // Name valid to for's "}"
     valSquared = userNum * userNum;
     cout << i << " squared: " << valSquared << endl;
  }

   // cout << "Last value: " << valSquared << endl; // ERROR

   return 0;
}
```


Animation content:

Static figure:
Begin C++ code:
#include <iostream>
using namespace std;

int main() {

  // int val1 = userNum;   // ERROR
 int userNum = 2;   // Name valid to main's "}"
 int newNum = userNum + 1;
 int i;

  for (i = 0; i < newNum; ++i) {
    int valSquared;   // Name valid to for's "}"
    valSquared = userNum * userNum;
    cout << i << " squared: " << valSquared << endl;
 }

  // cout << "Last value: " << valSquared << endl; // ERROR

  return 0;
}
End C++ code.

Animation captions:

1. userNum's scope is from the declaration to main's closing brace. Using userNum before the declaration would yield an "Undeclared name" compiler error.
2. A declaration can appear within any block (statements in braces {...}). valSquared is valid from the declaration to the closing brace.
3. valSquared is not valid outside that block.

PARTICIPATION ACTIVITY 4.12.2: Variable name scope.

Refer to the animation above.

1) userNum can be used in newNum's declaration.
   - True
   - False

2) If uncommented, userNum can be used in val1's declaration.
   - True
   - False

3) userNum can be used within the for loop's block of statements.
   - True
   - False

4) valSquared can be used within the for loop's block.
   - True
   - False

5) valSquared can be used in the for loop's loop variable update, such as replacing ++i by i = i + valSquared.
   - True
   - False

6) valSquared can be used just before main's return statement
   - True
   - False

For loop index

Programmers commonly declare a for loop's index variable in the for loop's initialization statement. That index variable's scope covers the other parts of the for loop, up to the for loop's closing brace. The reason is clear from the for loop's equivalent while loop code shown below, noting the braces around the equivalent code.

Table 4.12.1: Index variable declared in a for loop's initialization statement.

for loop	Equivalent while loop

```
for (int i = 0; i < 5; ++i)
{
   x = x + i;
}

x = x + i;  // ERROR
```

```
{
   int i = 0;
   while (i < 5) {
      x = x + i;
      ++i;
   }
}
x = x + i;  // ERROR
```


The approach of declaring a for loop's index variable in the for loop's initialization statement makes clear that the variable's sole purpose is to serve as that loop's index.

This material avoids declaring index variables in for loops

This material's authors have found that declaring all variables first, then using those variables in the rest of the code, can simplify learning for students. Thus, this material avoids late declarations of variables, including declaring index variables in for loops. With that said, declaring index variables in for loops is extremely common and considered good practice by many programmers, and thus is something to consider, if one can do so without confusion.

PARTICIPATION ACTIVITY 4.12.3: For loop index declared in loop's initialization statement.

Given the following for loop, determine whether index i's scope includes the indicated region.

```
(a)
for (int i = 0; (b); (c) ) {
   (d)
}
(e)
```

1) (a)

- Yes
- No

2) (b)

- Yes
- No

3) (c)

- Yes
- No

4) (d)

- Yes
- No

5) (e)

- Yes
- No

6) Suppose the above for loop is followed by a second for loop also with int i = 0 in the initialization statement. Will the compiler generate an error due to two declarations of i?

- Yes
- No

Common error

A common error is to declare a variable inside a loop whose value should persist across iterations. Below, the programmer expects the output to be 0, 1 (0+1), 3 (0+1+2), 6 (0+1+2+3), and 10 (0+1+2+3+4), but instead the output is just 0, 1, 2, 3, 4.

Figure 4.12.1: Common error: A variable declared within a loop block is (unexpectedly) re-initialized every iteration.

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

int main() {
   int i = 0;

   while (i < 5) {
      int tmpSum = 0;
      tmpSum = tmpSum + i; // Logic error: Sum is always
just i
      cout << "tmpSum: " << tmpSum << endl;
      i = i + 1;
   }

   return 0;
}
```

```
tmpSum:
0
tmpSum:
1
tmpSum:
2
tmpSum:
3
tmpSum:
4
```

PARTICIPATION ACTIVITY 4.12.4: Common error of a variable declared within a loop block being reinitialized every iteration.

Given the following code, indicate j's value at the specified point.

```
for (int i = 0; i < 5; ++i) {
   int j = 0;

   j = j * i;
}
```

1) At the end of iteration i = 0.

Check Show answer

2) At the end of iteration i = 1.

Check Show answer

3) At the end of iteration i = 2.

Check Show answer

4) After the loop terminates, can j be output? Type yes or no.

Check Show answer

4.13 C++ example: Salary calculation with loops

zyDE 4.13.1: Calculate adjusted salary and tax with deductions: Using loops.

A program may execute the same computations repeatedly.

The program below repeatedly asks the user to enter an annual salary, stopping when the user enters 0 or less. For each annual salary, the program determines the tax rate and computes the tax to pay.

1. Run the program below with annual salaries of 40000, 90000, and then 0.
2. Modify the program to use a while loop inside the given while loop. The new inner loop should repeatedly ask the user to enter a salary deduction, stopping when the user enters a 0 or less. The deductions are summed and then subtracted from the annual income, giving an adjusted gross income. The tax rate is then calculated from the adjusted gross income.
3. Run the program with the following input: 40000, 7000, 2000, 0, and 0. Note that the 7000 and 2000 are deductions.

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

Load default template...

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

int main() {
   const string SALARY_PROMPT = "\nEnter annual salar
   int annualSalary;
   int deduction;
   int totalDeductions;
   double taxRate;
   int taxToPay;

   cout << SALARY_PROMPT;
   cin >> annualSalary;
```



```
40000
90000
0
```

Run

A solution to the above problem follows. The input consists of three sets of annual salaries and deductions.

zyDE 4.13.2: Calculate adjusted salary and tax with deductions: Using loops (solution).

Load default template...

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

int main() {
   const string PROMPT_SALARY = "\nEnter annual salar
   const string PROMPT_DEDUCTION = "Enter a deductior
   int annualSalary;
   int oneDeduction;
   int totalDeductions;
   int adjustedSalary;
   double taxRate;
   int taxToPay;

   cout << PROMPT_SALARY << endl;
```



```
40000 3000 6000 0
90000 5000 0
60000 2000 1000 1450 0
```

Run

zyDE 4.13.3: Create an annual income and tax table.

A tax table shows three columns: an annual salary, the tax rate, and the tax amount to pay. The program below shows most of the code needed to calculate a tax table.

1. Run the program below and note the results.
2. Alter the program to use a for loop to print a tax table of annual income, tax rate, and tax to pay. Use starting and ending annual salaries of 40000 and 60000, respectively, and a salary increment of 5000.
3. Run the program again and note the results. You should have five rows in the tax table.
4. Alter the program to add user prompts and read the starting and ending annual incomes from user input.
5. Run the program again using 40000 and 60000, respectively, and the same salary increment of 5000. You should have the same results as before.
6. Alter the program to ask the user for the increment to use in addition to the starting and ending annual salaries.
7. Run the program again using an increment of 2500. Are the entries for 40000, 45000, 50000, 55000 and 60000 the same as before?

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;
   int startingAnnualSalary;
   int endingAnnualSalary;

   annualSalary = 0;
   startingAnnualSalary = 0;    // FIXME: Change the s
   endingAnnualSalary = 0;      // FIXME: Change the e

   // FIXME: Use a for loop to calculate the tax for
```


40000 60000 5000

Run

A solution to the above problem follows.

zyDE 4.13.4: Create an annual income and tax table (solution).

Load default template...

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

int main() {
   int annualSalary;
   double taxRate;
   int taxToPay;
   int startingAnnualSalary;
   int endingAnnualSalary;
   int incomeIncrement;

   cout << "Enter first annual salary for the table:
   cin >> startingAnnualSalary;

   cout << "Enter last annual salary for the table: "
```


40000 60000 2500

Run

4.14 C++ example: Domain name validation with loops

zyDE 4.14.1: Validate domain names.

A ***top-level domain*** (TLD) name is the last part of an Internet domain name like .com in example.com. A ***core generic top-level domain*** (core gTLD) is a TLD that is either .com, .net, .org, or .info. A ***second-level domain*** is a single name that precedes a TLD as in apple in apple.com

The following program uses a loop to repeatedly prompt for a domain name, and indicates whether that domain name consists of a second-level domain followed by a core gTLD. An example of a valid domain name for this program is apple.com. An invalid domain name for this program is support.apple.com because the name contains two periods. The program ends when the user presses just the Enter key in response to a prompt.

1. Run the program and enter domain names to validate. Note that even valid input is flagged as invalid.
2. Change the program to validate a domain name. A valid domain name for this program has a second-level domain followed by a core gTLD. Run the program again.

Load default template...

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

int main() {
   string coreGtld1;
   string coreGtld2;
   string coreGtld3;
   string coreGtld4;
   string inputName;
   string searchName;
   string theTld;
   bool isCoreGtld;
   // FIXME: Add variable periodCounter to count peri
   int periodPosition;   // Position of the period ir
```



```
apple.com
APPLE.COM
apple.comm
```

Run

A solution for the above problem follows.

zyDE 4.14.2: Validate domain names (solution).

Load default template...

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

int main() {
   string coreGtld1;
   string coreGtld2;
   string coreGtld3;
   string coreGtld4;
   string inputName;
   string searchName;
   string theTld;
   bool isCoreGtld;
   int periodCounter;
   int periodPosition;
```



```
apple.com
APPLE.COM
apple.comm
```

Run

4.15 LAB: Convert to reverse binary

This section has been set as optional by your instructor.

Write a program that takes in a positive integer as input, and outputs a string of 1's and 0's representing the integer in reverse binary. For an integer x, the algorithm is:

```
As long as x is greater than 0
   Output x % 2 (remainder is either 0 or 1)
   x = x / 2
```

Note: The above algorithm outputs the 0's and 1's in reverse order.

Ex: If the input is:


```
6
```

the output is:

```
011
```

6 in binary is 110; the algorithm outputs the bits in reverse.

LAB ACTIVITY | 4.15.1: LAB: Convert to reverse binary | 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.

4.16 LAB: Count characters

This section has been set as optional by your instructor.

Write a program whose input is a character and a string, and whose output indicates the number of times the character appears in the string. The output should include the input character and use the plural form, n's, if the number of times the

characters appears is not exactly 1.

Ex: If the input is:

```
n Monday
```

the output is:

```
1 n
```


Ex: If the input is:

```
z Today is Monday
```

the output is:

```
0 z's
```

Ex: If the input is:

```
n It's a sunny day
```

the output is:

```
2 n's
```

Case matters.

Ex: If the input is:

```
n Nobody
```

the output is:

```
0 n's
```

n is different than N.

539740.3879454.qx3zqy7

LAB ACTIVITY | 4.16.1: LAB: Count characters | 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.
```

4.17 LAB: Remove all non-alphabetic characters

This section has been set as optional by your instructor.

Write a program that removes all non alpha characters from the given input.

Ex: If the input is:

```
-Hello, 1 world$!
```

the output is:

```
Helloworld
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 4.17.1: LAB: Remove all non-alphabetic characters 0 / 10

main.cpp Load default template...

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
```

```
5
6    /* Type your code here. */
7
8    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.

4.18 LAB: Output range with increment of 5

This section has been set as optional by your instructor.

Write a program whose input is two integers, and whose output is the first integer and subsequent increments of 5 as long as the value is less than or equal to the second integer.

Ex: If the input is:

```
-15 10
```

the output is:

```
-15 -10 -5 0 5 10
```

Ex: If the second integer is less than the first as in:

```
20 5
```

the output is:

```
Second integer can't be less than the first.
```

For coding simplicity, output a space after every integer, including the last. End the output with a newline.

LAB ACTIVITY | 4.18.1: LAB: Output range with increment of 5 | 0 / 10

main.cpp — Load default template...

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5
6     /* Type your code here. */
7
8     return 0;
9  }
10
```

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.

4.19 LAB: Print string in reverse

This section has been set as optional by your instructor.

Write a program that takes in a line of text as input, and outputs that line of text in reverse. The program repeats, ending when the user enters "Done", "done", or "d" for the line of text.

Ex: If the input is:

```
Hello there
Hey
done
```

then the output is:

```
ereht olleH
yeH
```


LAB ACTIVITY | 4.19.1: LAB: Print string in reverse | 0 / 10

main.cpp — Load default template...

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5
6     /* Type your code here. */
7
8     return 0;
9  }
10
```

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.
```


©zyBooks 01/31/24 17:46 1939727
Rob Daglio
MDCCOP2335Spring2024