## Daniel Avila

Professor Ghadami

Intro to Programming 120

8 November 2019

## Fill in the blank 1-20

1. To **Increment** a value means to increase it by one.
2. To **Decrement** a value means to decrease it by one.
3. When the increment or decrement operator is placed before the operand (or to the operand’s right), the operator is being used in **Prefix** mode.
4. When the increment or decrement operator is placed after the operand (or to the operand’s right), the operator is being used in **Postfix** mode.
5. The statement or block that is repeated is known as the **Body** of the loop.
6. Each repetition of a loop is known as a(n) **Iteration**.
7. A loop that evaluates its test expression before each repetition is a(n) **Pretest** loop.
8. A loop that evaluates its test expression after each repetition is a(n) **Posttest** loop.
9. A loop that does not have a way of stopping is a(n) **Infinite** loop.
10. A(n) **Counter** is a variable that “counts” the number of times a loop repeats.
11. A(n) **Running** **Total** is a sum of numbers that accumulates with each iteration of a loop.
12. A(n) **Accumulator** is a variable that is initialized to some starting value, usually zero, and then has numbers added to it in each iteration of a loop.
13. A(n) **Sentinel** is a special value that marks the end of a series of values.
14. The **For** **Loop** is ideal for situations that require a counter.
15. The **Do**-**While** **Loop** always iterates at least once.
16. The **While** **Loop** and **For** **Loops** will not iterate at all if their test expressions are false to start with.
17. Inside the for loop’s parenthesis, the first expression is the Initialization, the second expression is the Test, and the third expression is the **Update**.
18. A loop inside another is called a(n) **Nested** **Loop**.
19. The **Break** statement causes a loop to terminate immediately.
20. The **Continue** statement causes a loop to skip the remaining statements in the current iteration.

## Algorithm Workbench: 33-34-35-36

1. while 🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪 do-while

|  |  |
| --- | --- |
| char doAgain = 'y';  int sum = 0;  cout << "This code will increment sum 1 or more times.\n";  while ((doAgain == 'y') || (doAgain == 'Y'))  {  sum++;  cout << "Sum has been incremented. Increment it again (y/n)? ";  cin >> doAgain;  }  cout << "Sum was incremented " << sum << " times.\n"; | char doAgain = 'y';  int sum = 0;  cout << "This code will increment sum 1 or more times.\n";  do {  sum++;  cout << "Sum has been incremented. Increment it again (y/n)? ";  cin >> doAgain;  } while ((doAgain == 'y') || (doAgain == 'Y'));  cout << "Sum was incremented " << sum << " times.\n"; |

1. do-while 🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪 while (no *if* statement)

|  |  |
| --- | --- |
| int number;  cout << "Enter an even number: ";  do  {  cin >> number;  if (number % 2 != 0)  cout << "Number must be even. Reenter number: ";  } while (number % 2 != 0); | int number;  cout << "Enter an even number: ";  cin >> number;  while (number % 2 != 0)  {  cout << "Number must be even. Reenter number: ";  cin >> number;  } |

1. while 🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪 for

|  |  |
| --- | --- |
| int count = 0;  while (count < 50)  {  cout << "count is " << count << endl;  count++;  } | for (int count = 0; count < 50; count++)  {  cout << "count is " << count << endl;  } |

1. for 🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪 while

|  |  |
| --- | --- |
| for (int x = 50; x > 0; x--)  {  cout << x << " seconds to go.\n";  } | int x = 50;  while (x > 0)  {  cout << x << " seconds to go.\n";  x--;  } |

## Predict the Output: 39-42

1. input 🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪 output

|  |  |
| --- | --- |
| int x = 1;  while (x < 10);  x++;  cout << x; | [blank]  while (x < 10);🡨  Because of the semi colon after the while condition. |

1. input 🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪 output

|  |  |
| --- | --- |
| int x = 1;  while (x < 10)  x++;  cout << x; | [10]  { statements }  Only displays 10 because there are no braces before the statements and after. It only runs the last line because with no braces, it only reads one line where it is true. |

1. input 🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪 output

|  |  |
| --- | --- |
| for (int count = 1; count <= 10; count++)  { cout << ++count << " "; //This is a bad thing to do!  } | [2 4 6 8 10]  With the initialization being 1, the statement adds 1 so then it becomes even, then the update adds another so now it is odd until the loop iterates again so it only displays even integers. |

1. input 🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪 output

|  |  |
| --- | --- |
| for (int row = 1; row <= 3; row++)  {  cout << "\n$";  for (int digit = 1; digit <= 4; digit++)  cout << '9';  } | [$9999  $9999  $9999]  The outer loop iterates first because its statement is outside of the inner loop and inner loop iterates next. |