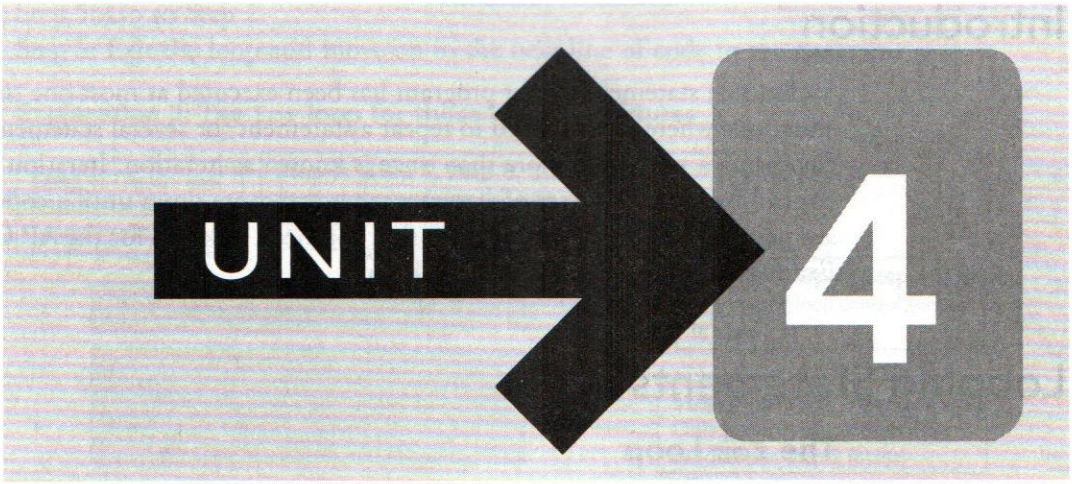



# Iteration

IN THIS UNIT

**Summary:** Iteration causes statements in your program to be repeated more than once. The two common iterative structures are the `for` loop and the `while` loop. Iteration is a fundamental programming structure and is used to solve many common tasks.


**Key Ideas**

- Looping structures allow you to repeat instructions many times.
- A `for` loop is used to repeat one or more instructions a specific number of times.
- A `while` loop is used when you don't know how many times you need to repeat a set of instructions but you know that you want to stop when some condition is met.
- Variables declared within a loop are only known within the loop.
- An infinite loop is a loop that does not end.
- A boundary error occurs when you don't execute a loop the correct number of times.

# Introduction

So far each statement in our program has been executed at most one time. There are many situations where we will need to repeat a statement (or several statements) more than once. Repeating a statement more than once is known as iteration. Iteration changes the flow of control by repeating a set of statements zero or more times until a condition is met. There are two types of iterative structures that you need to know for the AP Computer Science A Exam: the `while` loop and the `for` loop.

# Looping Statements

## The `for` Loop

Suppose your very strict music teacher catches you chewing gum in class and tells you to write out "I will not chew gum while playing the flute" 100 times. You decide to use your computer to do the work for you. How can you write a program to repeat something as many times as you want? The answer is a **loop**. A `for` loop is used when you know ahead of time how many times you want to do a set of instructions. In this case, we know we want to write the phrase 100 times.

**General Form for Creating a `for` Loop**

```
for (initialize loop control variable(LCV);condition using LCV; modify LCV)
{
   // instructions to be repeated
}
```

The `for` loop uses a **loop control variable** to repeat its instructions. This loop control variable is typically an `int` and is allowed to have a one-letter name like i, j, k, and so on. This breaks the rule of having meaningful variable names. Yes, the loop control variable is a rebel.

**Explanation 1: The `for` loop for those who like to read paragraphs . . .**

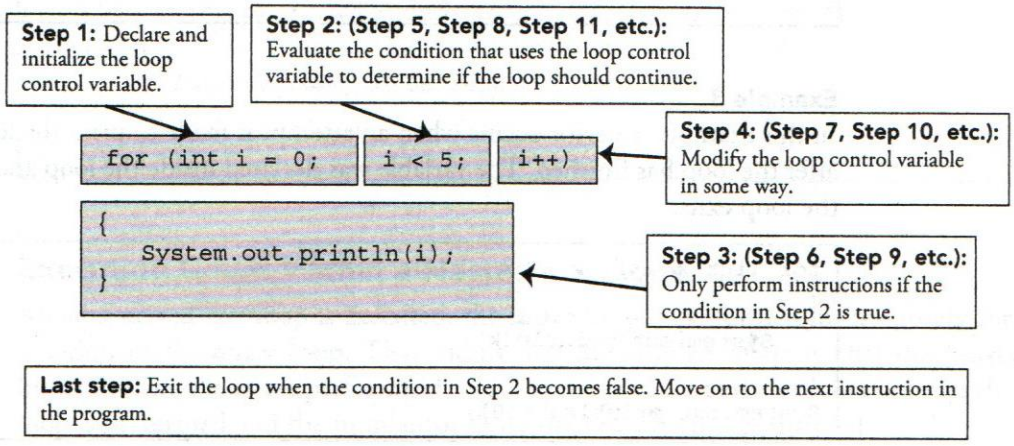
When a `for` loop starts, its loop control variable is declared and initialized. Then, a comparison is made using the loop control variable. If the result of the comparison is true, the instructions that are to be repeated are executed one time. The loop control variable is then modified in some way (usually incremented or decremented, but not always). Next, the loop control variable is compared again using the same comparison as before. If the result of this comparison is true, the loop performs another **iteration**. The loop control variable is again modified (in the same way that it was before), and this process continues until the comparison of the loop control variable is false. When this happens, the loop ends and we **exit (or terminate) the loop**. The total number of times that the loop repeats the instructions is called the number of iterations.

**Explanation 2: The `for` loop for those who like a list of steps . . .**

Step 1: Declare and initialize the loop control variable.
Step 2: Compare the loop control variable in some way.
Step 3: If the result of the comparison is true, then execute the instructions one time.
If the result of the comparison is false, then skip to Step 6.
Step 4: Modify the loop control variable in some way (usually increment or decrement, but not always).

Step 5: Go to Step 2.
Step 6: Exit the loop and move on to the next line of code in the program.

**Explanation 3: The `for` loop for those who like pictures . . .**




**The Loop Control Variable**

The most common approach when using a `for` loop is to declare the loop control variable in the `for` loop. If this is the case, then the loop control variable is only known inside the `for` loop. Any attempt to use the loop control variable in an instruction after the loop will get a compile-time error (cannot be resolved to a variable).

However, it is possible to declare a loop control variable prior to the `for` loop code. If this is the case, then the loop control variable is known before the loop, during the loop, and after the loop.

**Example 1**

A simple example of a `for` loop:

```
for (int i = 0; i < 3; i++)
{
   System.out.println("hello");
}
```

```
OUTPUT
hello
hello
hello
```

**Example 2**

The loop control variable is often used inside the loop as shown here:

```
for (int j = 9; j > 6; j--)
{
   System.out.print(j + "    ");
   System.out.println(10 - j);
}
```

```
OUTPUT
9       1
8       2
7       3
```

**Example 3**

In this example, an error occurs when an attempt is made to print the loop control variable after the loop has finished. The variable was declared inside the loop and is not known after the loop exits:

```
for (int k = 1; k <= 4; k++)
{
    System.out.println(k);
}
System.out.println(k*10);
```

```
Compile-time error: k cannot be resolved to a variable
```

**Example 4**

In this example, the loop control variable is declared in a separate instruction before the loop. This allows the variable to be known before, during, and after the loop:

```
int i;
for (i = 1; i < 12; i +=3)
{
    System.out.println(i);
}
System.out.println(i);
```

```
OUTPUT
1
4
7
10
13
```

***Fun Fact:*** *In an early programming language called Fortran, the letters i, j, and k were reserved for storing integers. To this day, programmers still use these letters for integer variables even though they can be used to store other data types.*

## The Nested `for` Loop

A `for` loop that is inside of another `for` loop is called a **nested for loop**. The **outer loop control variable** (OLCV) is used to control the outside loop. The **inner loop control variable** (ILCV) is used to control the inside loop. There is no limit to the number of times you can nest a series of `for` loops.

**General Form for a Nested `for` Loop**

```
for (initialize OLCV; condition using OLCV; modify OLCV)
{
   for (initialize ILCV; condition using ILCV; modify ILCV)
   {
      // instructions to be repeated
   }
}
```

## Execution Count Within a Nested `for` Loop

When a nested `for` loop is executed, the **inner loop** is performed in its entirety for every iteration of the **outer loop**. That means that the number of times that the instructions inside the inner loop are performed is equal to the product of the number of times the outer loop is performed and the number of times the inner loop is performed.

### Example

To demonstrate execution count for a nested `for` loop: The outer loop repeats a total of three times (once for i = 0, then for i = 1, and then for i = 2). The inner loop repeats four times (once for j = 1, then for j = 2, then for j = 3, and then for j = 4). The inner loop repeats (starts over) for every iteration of the outer loop. Therefore, the `System.out.println` statement executes a total of 12 times:

```
for (int i = 0; i < 3; i++)                  // the outer loop repeats 3 times
{
   for (int j = 1; j < 5; j++)               // the inner loop repeats 4 times
   {
      System.out.println(i + "  " + j);      // total of 12 iterations (3 * 4)
   }
}
```

```
OUTPUT
0    1
0    2
0    3
0    4
1    1
1    2
1    3
1    4
2    1
2    2
2    3
2    4
```

**Execution Count for a Nested `for` Loop**

To find the total number of times that the code inside the inner `for` loop is executed, multiply the number of iterations of the outer loop by the number of iterations of the inner loop.

```
for (int x = 4; x > 0; x--)          // the outer loop repeats 4 times
{
   for (int y = 9; y <= 15; y++)     // the inner loop repeats 7 times
   {
      // some instruction            // total of 28 iterations (4 * 7)
   }
}
```

## The `while` Loop

Consider the software that is used to check out customers at a store. The clerk drags each item the customer buys over the scanner until there aren't any more items to be scanned. *Viewing this through the eyes of a programmer*, I can suggest that a **while loop** is used to determine the final cost of all the items. The process repeats until there aren't any more items. Now, in contrast, if the clerk asked you how many items you had in your cart before he started scanning, then I would suggest that a `for` loop is being used to determine the final cost of all the items.

**General Form for a `while` Loop**

Recommended form: Repeat instructions (using curly braces):

```
while (condition)
{
   // one or more instructions to be repeated
}
```

The `while` loop repeats instructions just like the `for` loop, but it does it differently. Instead of having a loop control variable that is a number like the `for` loop, the `while` loop can use any kind of data type to control when it is to perform another iteration of the loop. I like to use the phrase *as long as* as a replacement for the word *while* when reading a `while` loop. In other words: *As long as the condition is true, I will continue to repeat the instructions.* It should be noted that a `for` loop can be rewritten into an equivalent `while` loop, and vice versa (but not as easily, so it isn't recommended). If the Boolean expression in the condition evaluates to false initially, then the loop body is not executed at all.

**Explanation 1: The `while` loop for those who like to read paragraphs . . .**

Declare and initialize some variable of any data type *before* the loop begins. Then compare this variable at the start of the loop. If the result of the comparison is true, the instructions inside the loop get executed one time. Then, modify the variable in some way. Next, compare the variable again in the same way as before. If the result is true, the instructions get executed one more time. This process continues until the result of the comparison is false. At this point, the loop ends and we exit the loop. You must make certain that the variable that is being compared is changed inside of the loop. If you forget to modify this variable, you may end up with a loop that never ends!

**Explanation 2: The `while` loop for those who like a list of steps . . .**

Step 1: Declare and initialize some variable of any data type.
Step 2: Compare this variable in some way.
Step 3: If the result of the comparison is true, then execute the instructions one time.
If the result of the comparison is false, then skip to Step 6.
Step 4: Modify the variable that is to be compared.
Step 5: Go to Step 2.
Step 6: Exit the loop and move on to the next line of code in the program.

**Example 1**

A simple example of a `while` loop:

```
int count = 0;
while (count < 5)
{
   System.out.println(count);
   count++;
}
```

```
OUTPUT
0
1
2
3
4
```

**Example 2**

Use a `while` loop to require a user to enter the secret passcode for a vault that contains a million dollars. Do not allow the user to exit the loop until they get it right. If they get it right the first time, do not enter the loop (do not execute any of the instructions inside the loop).

```
System.out.print("Enter the 4-digit secret passcode: ");
int secretPasscode = /* int provided by user */
while (secretPasscode != 1234)
{
   System.out.println("Sorry that is incorrect. Try again: ");
   secretPasscode = /* int provided by user */
}
System.out.println("Congratulations. You have access to the vault.");
```

```
OUTPUT
Enter the secret passcode: 9999
Sorry that is incorrect. Try again: 8888
Sorry that is incorrect. Try again:  1234
Congratulations. You have access to the vault.
```

**A Flag in Programming**

A **flag** in programming is a virtual way to signal that something has happened. Think of it as, "I'm waving a flag" to tell you that something important just occurred. Normally, `boolean` variables are chosen for flags. They are originally set to false, then when something exciting happens, they are set to true.

**Example 3**

Use a `while` loop to continue playing a game as long as the game is not over:

```
boolean gameOver = false;
while (!gameOver)
{
    // play the game
    // When a player wins, set gameOver = true
}
```

**Strength of the `while` Loop**

A strength of the `while` loop is that you don't have to know how many times you will execute the instructions before you begin looping. You just have to know when you want it to end.

## The Infinite Loop

If you make a mistake in a looping statement such that the loop never ends, the result is an **infinite loop**. You have to make sure that at some point in the loop body there is a change being made to the Boolean expression to make it false; otherwise, it will always be true and the loop will never end.

A loop must always have an "ITCH": Initialized variable, Tested variable, and CHanged variable.

```
for (initialize; test; change)
{
     // body of loop
}
```

```
initialize
while (test)
{
      // body of loop
      change
}
```

**Example 1**

Accidentally getting an infinite loop when using a `for` loop:

```
for (int i = 0; i < 10; i--)
{
    i++;
    // i will always revert back to 0 causing an infinite loop
}
```

**Example 2**
Accidentally getting an infinite loop when using a `while` loop:

```
int i = 0;
while (i < 10)
{
   // do something forever because you forgot to increment i
}
```

### Boundary Testing

A very common error when using a `for` loop or a `while` loop is called a **boundary error**. It means that you accidently went one number past or one number short of the right amount. This is known as an "off by one" error, or "obo." *The loop didn't perform the precise number of iterations.* This concept is commonly tested on the AP Computer Science A Exam. The only real way to prevent it is to hand-trace your code very carefully to make sure that your loop is executing the correct number of times.

**Goldilocks and the Three Looping Structures**

Always test that your `for` loops, nested `for` loops, and `while` loops are executing the correct number of times—not too many, but not too few. Just the right number of times. Hint: If you use:

```
for (int i = 0; i < maximum; i++)
```

then the loop will execute maximum number of times.

## Standard Algorithms

Let's take a look at some common tasks that can be done using loops that you might see on the AP Computer Science A Exam. But the tasks shown here are not the only things that you might see on the exam. You are expected to be able to read and/or write any task that involves anything that you have learned so far such as variables, conditionals, and loops for the exam.

**Example 1**
Find all the factors of an integer.

```
int number =    /*  some integer value  */
number = Math.abs(number);           // in case the number is negative
for (int i = 1; i <= number; i++)
{
   if (number % i == 0)          // if the remainder is 0, it is a factor
   {
      System.out.println(i + " is a factor");
   }
}
```

**Example 2**

Find the sum of all the digits in an integer.

```
int value = /*some integer value*/
int number = Math.abs(value);   // in case the number is negative
int digit, sum = 0;             // you need to initialize the sum to 0
while (number > 0)              // as long as you still have digits left
{
  digit = number % 10;          // mod 10 results in the ones digit
  sum += digit;                 // add that digit to the accumulated sum
  number /= 10;                 // dividing by 10 removes the ones digit
}
System.out.println("The sum of the digits of " + value + " is " + sum);
```

**Example 3**

Find the largest value in a list of numbers.

```
Scanner in = new Scanner(System.in);  // or some other way to get
                                      //     input from the user
System.out.println("Please enter a value. Use -99999 to end");
                                      //   -99999 used as sentinel value
double number = in.nextDouble();      // get the value from the user
double highest = number;              // initialize to the first number
                                      // entered
while (number != -99999)              // as long as the value entered
{                                     //       isn't the sentinel value
   if (number > highest)              // compare the value. If it's higher
   {                                  //      than the current highest
       highest = number;              // replace it
   }
   System.out.println("Please enter a value. Use -99999 to end");
                                      // does the user want to continue?
   number = in.nextDouble();          // get the next value
}
System.out.println("The highest value was  " + highest);
```

**Example 4**

Find the mean test score in a class of 25 students.

```
Scanner in = new Scanner(System.in);    // or some other way to get
                                        //     input from the user
int grade, sum = 0;                     // initialize the sum to 0
for (int i = 1; i <= 25; i++)
{       // since we know how many times, we can use a for loop
   System.out.print("Please enter student " + i + " test score...");
   grade = in.nextInt();                // gets the student grade
   sum += grade;                // adds that grade to the accumulated sum
}

int mean = (int) (sum / 25.0 + 0.5);    // if we want an integer mean,
                                    // double division must be used and
                                  // 0.5 added to it rounds properly

System.out.println("The mean score is " + mean);
```

**Example 5**

Count the number of spaces that are in a string.

```
String phrase = "AP Computer Science A Rocks!";
int numWords = 1;                              // initialize the count to 1
int posSpace = phrase.indexOf(" ");            // find the first space
while (posSpace != -1)       // as long as there is a space in the phrase
{
   numWords++;                                  // increment the word count
   phrase = phrase.substring(posSpace+1);     // the new phrase will be
                                              // everything after the space
   posSpace = phrase.indexOf(" ");             // find the next space
}
System.out.println("The number of words is " + numWords);
```

**Example 6**

Count the number of punctuation marks in a sentence.

```
String sentence = "A. P. Computer Science, Rocks!";
String punctuation = ".,:;?!";     // or whatever punctuation marks you
                                   //      want to search for
int count = 0;
String letter;
for (int i = 0; i < sentence.length(); i++)    // look through each letter
{
   letter = sentence.substring(i, i+1);        // get one letter at a time
   if (punctuation.indexOf(letter) >= 0) // if the letter is found in the
   {                                    // punctuation string, then add to
     count++;                           //      count
   }
}

System.out.println("The number of punctuation marks is " + count);
```

**Example 7**

Reverse the order of the characters in a string.

```
String sentence = "AP Computer Science Rocks";
String reverse = "", letter;
for (int i = sentence.length()-1; i >= 0; i--)    // start at the end of
{                                                 //      the sentence
   letter = sentence.substring(i, i+1);     // get one letter at a time
   reverse += letter;                 // concatenate the letter to the end
}

System.out.println("The reverse of " + sentence + " is " + reverse);
```

# › Rapid Review

### Programming Statements

- Curly braces come in pairs and the code they contain is called a block of code.
- Curly braces need to be used when more than one instruction is to be executed for `for` and `while` statements.
- The scope of a variable refers to the code that knows that the variable exists.
- Variables declared within a loop are only known within that loop.
- The `for` loop is used to repeat one or more instructions a specific number of times.
- The `for` loop is a good choice when you know exactly how many times you want to repeat a set of instructions.
- The `for` loop uses a numeric loop control variable that is compared and also modified during the execution of the loop. When the comparison that includes this variable evaluates to false, the loop exits.
- Variables declared within a `for` loop are only known within that `for` loop.
- A nested for loop is a `for` loop inside of a `for` loop.
- The number of iterations of a nested `for` loop is the product of the number of iterations for each loop.
- The `while` loop does not require a numeric loop control variable.
- The `while` statement must contain a conditional that compares a variable that is modified inside the loop.
- Choose a `while` loop when you don't know how many times you need to repeat a set of instructions but know that you want to stop when some condition is met.
- An infinite loop is a loop that never ends.
- Forgetting to modify the loop control variable or having a condition that will never be satisfied are typical ways to cause infinite loops.
- Variables declared within a `while` loop are only known within the `while` loop.
- A boundary error occurs when you don't execute a loop the correct number of times.

# › Review Questions

## Basic Level

**1.** Consider the following code segment.

```
int count = 1;
int value = 31;
while (value >= 10)
{
   value = value - count;
   count = count + 3;
}
System.out.println(value);
```

What is printed as a result of executing the code segment?

(A) `4`
(B) `9`
(C) `10`
(D) `13`
(E) `31`

**2.** Consider the following code segment.

```
int count = 5;
for (int i = 3; i < 7; i = i + 2)
{
   count += i;
}
System.out.println(count);
```

What is printed as a result of executing the code segment?

(A) 5
(B) 7
(C) 9
(D) 13
(E) 20

**3.** Consider the following code segment.

```
int incr = 1;
for (int i = 0; i < 10; i+= incr)
{
   System.out.print(i - incr + "    ");
   incr++;
}
```

What is printed as a result of executing the code segment?

(A) -1  0  2  5
(B) -1  0  3  7
(C) -1  1  4  8
(D) 0  2  5  9
(E) 0  1  3  7

**4.** Which of the following code segments will print exactly eight "$" symbols?

I.
```
for (int i = 0; i < 8; i++)
{
    System.out.print("$");
}
```

II.
```
int i = 0;
while (i < 8)
{
   System.out.print("$");
   i++;
}
```

III.
```
for (int i = 7; i <= 30 ; i+=3)
{
   System.out.print("$");
}
```

(A) I only
(B) II only
(C) I and II only
(D) I and III only
(E) I, II, and III

5. Consider the following code segment.

```
int a = /* value supplied within program */
int b = /* value supplied within program */
if (/* missing condition */)
{
   int result = a + b;
}
```

Which of the following should replace `/* missing condition */` to ensure that `a + b` will not be too large to be held correctly in result?

(A) `a + b < Integer.MAX_VALUE`
(B) `a + b <= Integer.MAX_VALUE`
(C) `Integer.MIN_VALUE + a <= b`
(D) `Integer.MAX_VALUE - a <= b`
(E) `Integer.MAX_VALUE - a >= b`

## Advanced Level

6. Consider the following code segment.

```
for (int j = 0; j < 10; j++)
{
   for (int k = 10; k > j; k--)
   {
      System.out.print("*");
   }
}
```

How many `"*"` symbols are printed as a result of executing the code segment?

(A) 5
(B) 10
(C) 45
(D) 55
(E) 100

7. Consider the following code segment.

```
boolean b1 = true;
boolean b2 = true;
int x = 7;
while (b1 || b2)
{
   if (x > 4)
   {
      b2 = !b2;
   }
   else
   {
      b1 = !b1;
   }
   x--;
}
System.out.print(x);
```

What is printed as a result of executing the code segment?

(A) 3
(B) 4
(C) 6
(D) 7
(E) Nothing will be printed; infinite loop.

**8.** Which of the following three code segments will produce the same output?

I.
```
int i = 1;
while (i < 12)
{
   System.out.print(i + " ");
   i *= 2;
}
```

II.
```
for (int i = 4; i > 0; i--)
{
   System.out.print((4 - i) * 2 + 1 + " ");
}
```

III.
```
for (int i = 0; i < 4; i++)
{
   System.out.print((int)Math.pow(2, i) + " ");
}
```

(A) I and II only
(B) II and III only
(C) I and III only
(D) I, II, and III
(E) All three outputs are different.

**9.** Consider the following code segment.

```
double count = 6.0;
for (int num = 0; num < 5; num++)
{
   if (count != 0 && num / count > 0)
   {
      count -= num;
   }
}
System.out.println(count);
```

What is printed as a result of executing the code segment?

(A) 0.0
(B) -4.0
(C) 5
(D) The program will end successfully, but nothing will be printed.
(E) Nothing will be printed. Run-time error: `ArithmeticException`.

**10.** Consider the following code segment.

```
int n = 0;
while (n < 20)
{
   System.out.print(n % 4 + " ");
   if (n % 5 == 2)
   {
      n += 4;
   }
   else
   {
      n += 3;
   }
}
```

What is printed as a result of executing the code segment?

(A) 0 3 2 1 0 0 3
(B) 0 0 3 2 2 1
(C) 0 3 2 1 0 0
(D) 0 0 3 2 2 1 0
(E) Many numbers will be printed; infinite loop.

**11.** Consider the following code segment.

```
double tabulate = 100.0;
int repeat = 1;
while (tabulate > 20)
{
   tabulate = tabulate - Math.pow(repeat, 2);
   repeat++;
}
System.out.println(tabulate);
```

What is printed as a result of executing the code segment?

(A) 20.0
(B) 6.0
(C) 19.0
(D) 9.0
(E) -40.0

**12.** During one iteration of the outer loop of the program segment below, how many times is the body of the inner loop executed?

```
for (int i = 1; i <= n-1; i++)
      for (int j = n; j >= i + 1; j--)
         // body of loop
```

(A) i + 1
(B) n – i
(C) n – i + 1
(D) i – n + 1
(E) n(i – 1)