| [Print](javascript:window.print()) | [Close](javascript:parent.window.close()) |

**Course Transcript**

Software Program Control Flow Fundamentals

**Using branching statements for selections**

| [1. IF Statements](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a03_it_enus/output/html/course_transcript.html#t5) |

| [2. CASE Statements](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a03_it_enus/output/html/course_transcript.html#t3) |

| [3. Solving a Problem using Branching Statements](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a03_it_enus/output/html/course_transcript.html#t28) |

| [4. Completing Selection Statements](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a03_it_enus/output/html/course_transcript.html#t7) |

**Programming Loops for Repetitions**

| [1. FOR Loops](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a03_it_enus/output/html/course_transcript.html#t11) |

| [2. WHILE Loops](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a03_it_enus/output/html/course_transcript.html#t13) |

| [3. Solving a Repetition Problem](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a03_it_enus/output/html/course_transcript.html#t15) |

| [4. Using Loops](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a03_it_enus/output/html/course_transcript.html#t17) |

IF Statements

Learning Objectives

*After completing this topic, you should be able to*

* *recognize the features of IF statements*
* *recognize how programming languages process IF statements*

**1. The features of IF statements**

Structured programming includes three control structures – repetition, sequence, and selection. The repetition control structure represents the process whereby a series of commands are executed until specified criteria are met. The sequence control structure represents the order in which statements in a program are executed. The selection control specifies when and how a statement is chosen to execute, depending on what the programmer requires.

In code, IF statements are part of the selection control structure. Statements like these, which are part of the selection control structure, are also known as *conditional statements*.

Programmers use IF statements to select which statements to execute based on whether conditions are true or false.  
  
IF statements come in a variety of forms. The most basic form is IF...THEN, which instructs a program to check whether a specified condition is true before running a specified statement block.

If the condition in an IF...THEN statement is met - is true - the statement block associated with the THEN keyword will be executed. If the condition isn't met – is false – the statement block associated with the THEN keyword is ignored and the program moves to the next line of code after the IF..ENDIF structure.

Syntax

IF [condition is true] THEN  
    [statement block]  
END IF

Note

*In the BASIC language, the IF structure is terminated with the END IF statement.*

An IF statement compares one value or variable to another to determine whether a condition is true or false. For example, it may check whether the amount a customer spends is greater than $1000, and if it is, will apply a discount of 5% because the condition is true.

Code

IF purchases > 1000 THEN  
   total = purchases \* 0.95  
END IF

This makes the statement a binary decision. Either the statement is true, with a binary value of 1, or it's false, with a binary value of 0.

In IF statements, you can use six different comparison operators to compare items:

**=**

The equal sign (=) is the equals operator and is used to evaluate whether two values are the same. For example, 5 = 9 results in false.  
  
The equals operator can also be indicated by two equal signs (==), as is the case in "squiggly bracket" languages such as C and Java.

**<**

The less than operator is indicated by the less than sign (<) and is used to evaluate whether one value is less than another. For example, the statement 5 < 9 results in true.

**>**

The greater than operator is indicated by the greater than sign (>) and is used to evaluate whether one value is higher than another. For example, 5 > 9 results in false.

**<=**

The less than or equal to operator is indicated by the less than sign combined with the equal sign (<=) and is used to evaluate whether one value is equal to or lower than another value. For example, 5 <= 9 results in true.

**>=, and**

The greater than or equal to operator is indicated by the greater than sign combined with the equal sign (>=) and is used to evaluate whether one value is equal to or higher than another value. For example, 5 >= 9 results in false.

**<>**

The not equal to operator is indicated by the less than and greater than signs together (<>) and is used to evaluate whether one value is not equal to another value. Whereas 5 = 9results in false, 5 <> 9 results in true.  
  
In "squiggly bracket" languages such as C and Java, the not equal to operator is indicated by !=.

A selection condition can compare multiple factors to another factor, as in the expression (a+b)>c.

Consider an example of an IF statement written in the BASIC programming language. It will print a specified message on the receipts of customers only if they spend $1000 or more at a store.

Code

IF client\_purchases >= 1000 THEN  
   PRINT "We value your generous patronage."  
END IF   
PRINT client\_name

If a customer spends less than $1000, the program ignores the code after the THEN statement and prints only the client's name. In other words, when the IF statement is false, the program will continue to run the code after the END IF keyword, which marks the end of the IF...THENstatement.

You can include an ELSE clause to run alternative code if the condition in an IF statement evaluates as false. This type of statement is referred to as an IF...THEN...ELSE statement.

Graphic

*In the image there is a question box which asks :Is condition 1 met? Then there are two branches – the top branch will execute if the value is true or the bottom branch will execute if the statement is false.*

Syntax

IF [condition is true]       
    THEN       
        [statement1 block]      
    ELSE       
        [statement2 block]   
END IF

If the condition in the IF statement is met - is true, statement1 block will run. If the condition isn't met – is false, statement2 block – included after the ELSE keyword – will run.

Syntax

IF [condition is true]       
    THEN       
        [statement1 block]      
    ELSE       
        [statement2 block]   
END IF

If you manage an online store, you might want to let your customers know when a specific item is in stock. You can create an IF...THEN...ELSE statement that checks if there is stock of an item and then displays a different message depending on whether the item is or isn't available.

Code

IF Stock >= 1  THEN  
   PRINT "Stock available"  
ELSE  
   PRINT "Out of stock"  
END IF

Note

*The code shows a BASIC language example of the IF…THEN…ELSE construct.*

This is an example of an IF...THEN...ELSE statement written in Pascal. It specifies that if the price of an item plus tax is equal to or greater than 120, a program must execute the first WRITELNcommand. If the sum of the price and the tax is less than 120, the second WRITELN command will execute.

Code

if (price+tax) >= 120 then  
    writeln ('total price is equal to or more than 120')  
else  
    writeln ('total price is less than 120');

This example of an IF...THEN...ELSE statement is written in C. It specifies that if a client's balance is negative, a computer program must print the message "overdrawn." If the balance is positive, the program will print "in credit."

Code

if (balance < 0)   
{      
    printf("overdrawn \n");  
}   
else  
{      
    printf("in credit \n");  
}

Note

*Note that IF statements written in C don't end with the END IF keyword.  
  
The \n used in the printf statement is called an escape sequence - after printing something to the screen you usually want to print something on the next line. If there is no \n then a next printfcommand will print the next string on the same line.*

An IF statement that doesn't include an ELSE clause is also sometimes referred to as a *null* ELSEstatement.

Code

IF [purchases are more than $1000] THEN  
   [apply a discount of 5%]  
END IF

Question

Which descriptions of IF statements are correct?

**Options:**

1. An IF statement tells a program which code to execute based on whether a condition is true or false
2. The statement block in an IF...THEN statement will execute only if a specific condition is met
3. An IF...THEN...ELSE statement will run alternative code if a specified condition isn't met
4. All statement blocks in an IF...THEN...ELSE will run if a condition is met
5. A null ELSE statement is a statement that includes an ELSE command

Answer

***Option 1:****Correct. IF statements are part of the selection control structure. They enable a program to select code to execute based on whether a condition is true or false.*

***Option 2:****Correct. In an IF...THEN statement, the statement block after the THEN keyword runs only if the condition specified after the IF keyword is met.*

***Option 3:****Correct. If the condition in an IF...THEN...ELSE statement isn't met, the statement block after the ELSE command will run.*

***Option 4:****Incorrect. If the specified condition is met, only the first statement block in an IF...THEN...ELSE statement will run.*

***Option 5:****Incorrect. A null ELSE statement is an IF statement that doesn't include an ELSEstatement. It's the same as a simpleIF...THEN statement.*

**Correct answer(s):**

1. An IF statement tells a program which code to execute based on whether a condition is true or false  
2. The statement block in an IF...THEN statement will execute only if a specific condition is met  
3. An IF...THEN...ELSE statement will run alternative code if a specified condition isn't met

**2. Processing IF statements**

Sometimes you may need to include more than one condition in an IF statement. This is known as a compound condition.

Graphic

*In this diagram representing an IF statement, a compound condition is created, which first tests Requirement 1 AND Requirement 2 then another AND is used to join the condition to a second condition which tests Requirement 3 OR Requirement 4. Then based of the results of the compound condition either the true or false branch will be executed.*

Compound conditions use the logical operators AND or OR.

The AND operator joins two conditions so that an expression evaluates as true only if both those conditions are met. If only one or other of the conditions is met, the expression evaluates as false.

The OR operator specifies that an expression is true if either of two conditions is true. So if one condition is true and the other is false, the result of the expression is true.

Another operator that can be used in an IF statement is the NOT operator. It is used to specify that an expression is true only if a condition is actually false.

Consider an IF...THEN...ELSE statement with a condition that uses the AND operator to check whether a cup of coffee meets each of four conditions.

Code

IF coffee AND cream AND sugar AND (Jamaican OR Javanese)   
   THEN   
       accept coffee  
   ELSE  
       decline coffee  
END IF

The cup of coffee has to have cream, coffee, and sugar, and to be either Jamaican or Javanese to be accepted.

Graphic

*The relevant code is:  
coffee AND cream AND sugar AND (Jamaican OR Javanese)  
Here is the code again.*

Code

IF coffee AND cream AND sugar AND (Jamaican OR Javanese)   
   THEN   
       accept coffee  
   ELSE  
       decline coffee  
END IF

The final condition uses the OR operator to specify that the coffee can be either Jamaican or Javanese.

Graphic

*The relevant code is:  
(Jamaican OR Javanese)  
Here is the code again.*

Code

IF coffee AND cream AND sugar AND (Jamaican OR Javanese)   
   THEN   
       accept coffee  
   ELSE  
       decline coffee  
END IF

Consider an IF...THEN...ELSE statement written in Pascal to check air pressure and wind direction.

Code

if ((air\_pressure > 1000) and (wind = 'nw')) then  
   writeln ('It is likely to rain.')  
else  
   writeln ('It probably won't rain.') ;

The statement is considered true only if the conditions for air pressure and for wind direction are both met. If both conditions are met, the message "It is likely to rain" will display. If one or both conditions fail, the message "It probably won't rain" will display.

A nested IF statement is an IF statement within another IF statement. You can use this type of statement to specify multiple, separate conditions.

Graphic

*In the image there is a question box which asks: Is condition 1 met? Then there are two branches – the top branch will execute if the value is true or the bottom branch will execute if the statement is false.  
  
The true branch then has a question box which asks: Is condition 2 met? If condition 2 is met, then a second set of true conditions will execute. If the condition is not met, then a different set of conditions will execute.*

Say you need to write a C++ program that checks the overdraft charges for savings and checking accounts. The program first checks whether an account has a positive balance. If it does, the first printf statement will run.

Graphic

*The relevant code is:  
if (balance >= 0)   
{      
   printf ("Account in credit");*

Code

if (balance >= 0)   
{      
    printf ("Account in credit");  
}   
else  
{   
    if (account\_type == 'c')   
    {       
        printf ("Checking account overdraft charge is $15 \n");  
    }  
    else   
    {  
        printf ("Savings account overdraft charge is $12 \n");  
    }  
}

If the account has a negative balance, the program needs to check what type of account it is. So the nested if statement after the else keyword will run.

Graphic

*The relevant code is:   
else  
{   
   if (account\_type == 'c')*

Code

if (balance >= 0)   
{      
    printf ("Account in credit");  
}   
else  
{   
    if (account\_type == 'c')   
    {       
        printf ("Checking account overdraft charge is $15 \n");  
    }  
    else   
    {  
        printf ("Savings account overdraft charge is $12 \n");  
    }  
}

The program will then check the type of account that has the overdraft. If the account type is "c", the first printf statement in the nested if statement will run. If the account type isn't "c", the second printf statement will run.

Graphic

*The relevant code is:  
       printf ("Checking account overdraft charge is $15 \n");  
   }  
   else   
   {  
       printf ("Savings account overdraft charge is $12 \n");  
   }  
}*

Code

if (balance >= 0)   
{      
    printf ("Account in credit");  
}   
else  
{   
    if (account\_type == 'c')   
    {       
        printf ("Checking account overdraft charge is $15 \n");  
    }  
    else   
    {  
        printf ("Savings account overdraft charge is $12 \n");  
    }  
}

Consider a BASIC program that calculates interest earned on positive credit card balances. It uses a nested IF statement to determine the interest calculated for three types of credit cards.

Code

IF (card\_type = "gold") THEN   
    IF (account\_type = "business") THEN   
        print "Interest calculated at 6%"   
    ELSE   
        print "Interest calculated at 5%"   
    END IF   
ELSE   
    print "Interest calculated at 4%"   
END IF

This statement first checks the type of card. If it's a gold card, the account will earn 5% or 6% interest, depending on whether it's a standard gold card or a business gold card. If it isn't a gold card, the account will earn 4% interest.

Code

IF (card\_type = "gold") THEN   
    IF (account\_type = "business") THEN   
        print "Interest calculated at 6%"   
    ELSE   
        print "Interest calculated at 5%"   
    END IF   
ELSE   
    print "Interest calculated at 4%"   
END IF

If the card is a gold card, the first nested IF statement is run. The nested IF statement determines if the account is a business account. If it is then the interest is set to 6%. If the account isn't a business account, the ELSE statement is run, which sets the interest at 5%.

Graphic

*The relevant code is:   
IF (card\_type = "gold") THEN   
   IF (account\_type = "business") THEN   
       print "Interest calculated at 6%"   
   ELSE   
       print "Interest calculated at 5%"   
   END IF*

Code

IF (card\_type = "gold") THEN   
    IF (account\_type = "business") THEN   
        print "Interest calculated at 6%"   
    ELSE   
        print "Interest calculated at 5%"   
    END IF   
ELSE   
    print "Interest calculated at 4%"   
END IF

If the card isn't a gold card, the nested IF statement is skipped. The interest is then set at 4%.

Graphic

*The relevant code is:   
ELSE   
   print "Interest calculated at 4%"   
END IF*

Code

IF (card\_type = "gold") THEN   
    IF (account\_type = "business") THEN   
        print "Interest calculated at 6%"   
    ELSE   
        print "Interest calculated at 5%"   
    END IF   
ELSE   
    print "Interest calculated at 4%"   
END IF

Question

You've written a simple program in C that uses IF statements.  
  
What does the nested IF statement accomplish?

**Code**  
if (numDozens < 12)  
{  
   numDozens = numDozens + 1;  
   gross = gross + 1;  
   if (countToADozen < 12)  
   {  
       countToADozen = countToADozen + 1;  
   } else  
   {  
       num = num + 1;  
       cout << numDozens << " dozen.\n";  
       countToADozen = 0;  
   }  
} else  
{  
   cout << "12 dozen.\n";  
   done = true;  
   cout << "TOTAL = " << gross << "\n";  
   cout << "One gross has been counted\n";  
}

**Options:**

1. It will increment the variable countToADozen by 1 unless the value of countToADozen equals 12 or higher
2. It will increment the variable numDozens by 1 unless the value of numDozens equals 12
3. It will increment the variable countToADozen by 1 if the value of countToADozen is greater than 12
4. It will set the value of countToADozen to 0 until the value of numDozens is greater than 12

Answer

***Option 1:****Correct. Unless the value of countToADozen equals 12 or higher, 1 will be added to the value for countToADozen. If the value of countToADozen is equal to 12, the nested IFstatement will break.*

***Option 2:****Incorrect. The nested IF statement is used to determine the value of countToADozen, not numDozens.*

***Option 3:****Incorrect. If the value of countToADozen is equal to 12, the nested IF statement will break and control will pass back to the first IF statement.*

***Option 4:****Incorrect. countToADozen will be set to 0, only if the value of countToADozen is already 12 or more when the nested IF statement starts executing.*

**Correct answer(s):**

1. It will increment the variable countToADozen by 1 unless the value of countToADozen equals 12 or higher

**3. Summary**

Programmers use IF statements to select which code statements to execute based on whether specified conditions are met. In an IF...THEN statement, a statement block after the THENkeyword executes only if the condition is true. An IF...THEN...ELSE statement works in the same way, but if the condition is false, a statement block after the ELSE keyword is executed.  
  
You can use the AND and OR logical operators in IF statements to create compound conditions. You can also use the NOT operator to specify that a statement block must execute if a condition is false. You can use nested IF statements to include multiple conditions and associated statement blocks.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a03_it_enus/output/html/course_transcript.html#top)

CASE Statements

Learning Objective

*After completing this topic, you should be able to*

* *identify the features of CASE statements*

**1. How to use CASE statements**

Although an IF statement is useful for making simple decisions, using IF statements for more than two options can become unnecessarily complicated.

Graphic

*In the diagram, when processing starts, an IF condition is tested. If it's false, the ELSE code executes and the program ends.   
  
If the condition is false, then another IF condition is tested. If this condition is false, the ELSE code executes and the program ends.  
  
If the condition is true, then the true code statements execute and the program will end.*

Nested IF statements can be used to deal with code that requires three or more options. However, the more options there are, the harder it becomes to understand and follow multiple nested IFstatements. This can make it difficult to debug code.

You can use CASE statements to replace complicated nested IF statements.

Syntax

SELECT CASE variable   
CASE value\_1  
  first statement block   
CASE value\_2  
  second statement block  
CASE value\_x  
  xth statement block  
CASE ELSE  
  optional ELSE clause/statement block   
END SELECT

A CASE statement consists of comparison options in a list, only one of which can be valid. Each option condition is unique and has an associated statement block. If a particular option is matched to the case variable, its associated statement block is executed – or run.

Graphic

*In this diagram, there is a Start command, then there are three boxes, one below the other, with conditions for testing. Each of these conditions is tested and will execute code according to the result of the condition. After a condition is met the CASE statement ends. If no condition is met, it will default to the OTHER block.*

Each CASE option can evaluate as many expressions as necessary. The final CASE ELSEstatement caters for all other possibilities not handled by a specific option.

In this example, if a value falls within the range defined for *value\_1*, the first statement block will run. If a value falls within the range defined for *value\_2*, the second statement block will run. The same applies to *value\_x* and the *xth\_statement* block.

Syntax

SELECT CASE *variable*CASE *value\_1                 
  f*i*rst statement block*CASE *value\_2*  *second statement block*CASE *value\_x*  *xth statement block*END SELECT

You might use a selection statement, for example, to define student's grades based on their test results. The code for doing this can be written using one of two methods.

You can use IF statements, which use Boolean variables, such as IF (result = “F”), to determine if statements are true or false. This example is correct, but is lengthy and hard to follow.

Code

IF result = "F"  
  THEN   
    PRINT "You got an F"  
END IF  
IF result = "D"  
  THEN   
    PRINT "You got a D"  
END IF  
IF result = "C"  
  THEN   
    PRINT "You got a C"  
END IF  
IF result = "B"  
  THEN   
    PRINT "You got a B"  
END IF  
IF result = "A"  
  THEN   
     PRINT "Good job! You got an A"  
END IF

Alternatively, you can use a CASE statement, which is much shorter and easier to follow. Each CASE statement must have an option for every possible result. Each result is checked to determine which range it falls into. If a result falls within a specific range, the statement associated with that range is executed.

Code

SELECT CASE result  
CASE “F”  
  PRINT "You got an F"  
CASE “D”  
  PRINT "You got a D"  
CASE “C”  
  PRINT "You got a C"  
CASE “B”  
  PRINT "You got a B"  
CASE “A”  
  PRINT "Good job! You got an A"  
END SELECT

CASE statements aren't limited to checking that a variable matches a single value. You can check for multiple values in a case line using the SELECT CASE statement.

Code

SELECT CASE Result  
CASE “F”  
  PRINT “Failure is not an option”  
CASE “D”, “C”, “B”  
  PRINT “Try harder next time”  
CASE “A”  
  PRINT “Excellent work!”  
END SELECT

For example, you can write code that prints different messages depending on the result. In this example, if a learner gets a B, C, or D, the message "Try harder next time" will print. This demonstrates how multiple conditions can be incorporated into one CASE statement.

You can also check for number variables using a SELECT CASE statement. If you have a list of numbers in sequence, you can specify the number for each case.

Graphic

*The relevant code is  
CASE 1,2,3,4,5,6,7,8,9,10,11*

Code

SELECT CASE Dozen  
CASE 1,2,3,4,5,6,7,8,9,10,11  
  Result = “Less than a dozen”  
CASE 12  
  Result = “One dozen”  
CASE 13  
  Result = “A baker’s dozen”  
END SELECT

You can also write a list of sequential numbers using the TO command. For example, you have 11 potential results for the first CASE statement. Instead of specifying each CASE number, you can simply list them as CASE 1 TO 11.

Code

SELECT CASE Dozen  
CASE 1 TO 11  
  Result = “Less than a dozen”  
CASE 12  
  Result = “One dozen”  
CASE 13  
  Result = “A baker’s dozen”  
END SELECT

You can use comparison operators, such as >, <, or <>, to expand the ways that the CASEstatement can be used.

Code

SELECT CASE Age  
CASE IS < 12  
  ageStatus = "Pre-teen"  
CASE 13 TO 19  
  ageStatus = "Teenager"  
CASE 20  
  ageStatus = "Unsure"  
CASE IS > 20  
  ageStatus = "Adult"  
END SELECT

In many programming languages, you can use comparison operators by using the IS or TOkeywords. The IS keyword checks the tested variable against the expression listed after it. The TOkeyword checks the tested variable using a range of values.

Code

SELECT CASE Age  
CASE IS < 12  
  ageStatus = "Pre-teen"  
CASE 13 TO 19  
  ageStatus = "Teenager"  
CASE 20  
  ageStatus = "Unsure"  
CASE IS > 20  
  ageStatus = "Adult"  
END SELECT

Note

*Curly bracket languages, like C and C++, don't allow you to use comparison operators in a switchstatement.*

In this SELECT CASE example, a different status message displays depending on the age group into which the value of the Age variable falls.

When you've created a range of CASE statements and need to use a variable that doesn't fit into any of the specified ranges, you can use the ELSE statement.

In this example, a final case determines that if the value of Age is 65, the corresponding status is retired.

Graphic

*The relevant code is:  
CASE 65  
 ageStatus = “Retired”*

Code

SELECT CASE workStatus  
CASE IS < 12  
  ageStatus = "Can’t work"  
CASE 13 TO 19  
  ageStatus = "Student"  
CASE 20 TO 64  
  ageStatus = "Time to find a job"  
CASE 65  
  ageStatus = "Retired"  
END SELECT

In another example, the ELSE command defines the status of anyone who falls outside of the defined categories, which cover people up to the age of 64. If someone is 65 years or older, the ELSE command will run and the status will be set as retired.

Graphic

*The relevant code is:  
CASE ELSE  
 ageStatus = “Retired”*

Code

SELECT CASE workStatus  
CASE IS < 12  
  ageStatus = "Can’t work"  
CASE 13 TO 19  
  ageStatus = "Student"  
CASE 20 TO 64  
  ageStatus = "Time to find a job"  
CASE ELSE  
  ageStatus = "Retired"  
END SELECT

Question

Identify the features of CASE statements.

**Options:**

1. They can be used to replace complicated nested IF statements
2. They work by choosing a single option from a list of options
3. They can only evaluate a limited number of expressions
4. They need to cater for every possible result

Answer

***Option 1:****Correct. Nested IF statements can become unnecessarily complicated when you're dealing with multiple options. You can use a CASE statement in this situation to replace the IF statements.*

***Option 2:****Correct. The option that's chosen determines which statement block will run.*

***Option 3:****Incorrect. A CASE statement can be used to evaluate as many expressions as necessary.*

***Option 4:****Incorrect. In many programming languages, CASE statements don't have to have options for every possible result.*

**Correct answer(s):**

1. They can be used to replace complicated nested IF statements   
2. They work by choosing a single option from a list of options

**2. Implementing CASE statements**

Not all programming languages use the CASE syntax. Instead they may use different syntax to serve the same function.

Say you want to print descriptions of three comparison operators – <, >, and = – using both Pascal and C.

Pascal uses the CASE statement to display the description of each operator.

Code

case comparator of       
    '<' : writeln ("less than");       
    '>' : writeln ("greater than");       
    '=' : writeln ("equal");  
else          
        writeln ("invalid");  
end

C uses the switch and break statements to select and display a description of a specific comparison operator.

Code

switch (comparator)  
{  
case '<': printf ("less than");      
    break;  
case '>': printf ("greater than");  
    break;  
case '=': printf ("equal");      
    break;  
default: printf ("invalid");  
}

Keyboard Sequence

*By convention, most keywords in C are written using lowercase.*

Like SELECT CASE, the switch statement is used to select a result from a set of options.

In languages like C, the break statement ends the execution of the switch block.

The switch statement processes case statements until it finds a match. After a case statement runs, the break statement halts further processing of case statements.   
  
In this example, if the value is the greater than operator (>), the printf command will print "greater than." The break command will then halt processing of any other commands in the switch statement, and processing will resume after the closing brace (}).

Code

switch (comparator)  
{  
case '<' : printf ("less than");      
    break;  
case '>' : printf ("greater than");  
    break;  
case '=' : printf ("equal");      
    break;  
default : printf ("invalid");  
}

Question

Which line of code executes when the value of ageStatus is 70?

**Code**  
SELECT CASE ageStatus  
CASE IS < 12  
  workStatus = “Can’t work”  
CASE 13 TO 19  
  workStatus = “Student”  
CASE 20 TO 64  
  workStatus = “Time to find a job”  
CASE ELSE  
  workStatus = “Retired”  
END SELECT

**Options:**

1. workStatus = “Retired”
2. workStatus = “Time to find a job”
3. workStatus = “Student”
4. workStatus = “Can’t work”

Answer

***Option 1:****Correct. If a value falls outside the ranges defined by case statements, as it does if ageStatus is equal to any value greater than 64, the code after the else keyword will execute.*

***Option 2:****Incorrect. This code will execute only if the value of ageStatus falls in the range from 20 to 64. In this example, the value of the variable is 70.*

***Option 3:****Incorrect. This code will execute only if the value of ageStatus falls in the range from 13 to 19.*

***Option 4:****Incorrect. This code will execute only if the value of ageStatus is less than 12.*

**Correct answer(s):**

1. workStatus = “Retired”

**3. Summary**

Instead of using potentially complex nested IF statements to handle three or more options in code, you can use CASE statements. Using this type of statement, you can provide a list of options and instruct a program to run a statement block associated with whichever option applies. In many programming languages, you use the keywords SELECT CASE at the beginning of a range of CASE statements to identify the variable being checked.  
  
Some programming languages use different syntax for the same purpose as SELECT CASE. For example, C uses switch statements, with break statements to halt processing once an option has been selected.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a03_it_enus/output/html/course_transcript.html#top)

Solving a Problem using Branching Statements

Learning Objectives

*After completing this topic, you should be able to*

* *identify the steps in solving a basic programming problem*
* *complete branching statements in C code*

**1. Solving a basic programming problem**

To simplify the programming process, which can be defined as the process of solving a programming problem, you can follow four main steps:

* analyze the problem, which is typically captured in a problem statement
* create a defining table
* create an algorithm in pseudocode, and
* write the algorithm code

To understand the problem completely, you might need to start by doing some research to put the problem in context.

Once you fully understand the problem, you can define its functional requirements by analyzing it in terms of its input, processing, and output elements.

The second step is to create a defining table that determines which data the program should process and what the result should be.

After identifying the input, process, and output elements, you can compile a table that lists the elements under the relevant headings.

Graphic

*A table includes Input, Process, and Output columns.*

Input and output elements are generally nouns in a problem statement, and processes are identified using verbs. It's processes that will transform the input into the output.

Every input element has an associated process to acquire the data associated with it. And every output element has a process to display or otherwise act on the data that has been processed.  
  
Some elements may not be explicitly mentioned in the problem statement, so you need to determine these from the context and the expected results.

Graphic

*In the first line of this diagram an Input flows into a process and on the second line a Process flows to an Output.*

There isn't necessarily a one-to-one mapping of processes to input and output elements. This is because many different procedures or functional elements may be required to act on a single input element or to produce a single output element.

Graphic

*In this diagram there are 3 Inputs, the first two flows into one process and the 3rd flows into one process. The two processes then flow out into one Output.*

In large modern object-oriented programs, such as those created using C++ or Java, hundreds of functional elements known as classes may be needed to model and implement the required behavior.

Graphic

*A large circle is in the center and four smaller circles are attached to it.*

The third step is to create an algorithm in pseudocode that outlines how the program will function. This should take the form of a detailed, ordered set of instructions that have to be carried out. But it doesn't implement these in any specific programming language.

Instead of using a specific programming language, a pseudocode algorithm uses clear and precise language to describe the instructions that have to be carried out, and in what order.

Code

IF x > y THEN      
   PRINT "X is: "; x   
END IF

The fourth and final step is to convert the pseudocode into a programming language. Pseudocode algorithms should be written in such a way that they can be encoded into any programming language.

Each programming language has different strengths and weaknesses, so you may have to consider the purpose of the program you're designing before choosing the language you'll use to create it.

Question

Sequence the steps for solving a programming problem.

**Options:**

1. Analyze the problem statement
2. Create a defining table
3. Create an algorithm in pseudocode
4. Write the algorithm in a programming language

Answer

**Correct answer(s):**

**Analyze the problem statement is ranked**

The first step is to create and analyze a problem statement, defining its input, processing, and output elements.

**Create a defining table is ranked**

After analyzing the problem statement, you compile a table that defines the relevant input, processing, and output elements.

**Create an algorithm in pseudocode is ranked**

After compiling a defining table you create an algorithm in pseudocode. This involves writing a detailed, ordered set of instructions that the program will have to execute.

**Write the algorithm in a programming language is ranked**

As the last step, you convert the pseudocode you've created into a programming language that's suitable given the purpose of the program.

**2. Branching statements in C code**

You work for a theater that's developing a program to manage bookings and ticket sales.

You've been asked to write the part of the program that calculates the total price of tickets bought by customers. The theater has three sections, each represented by a code, and each section has a limited number of seats. When tickets are requested, the program must check seat availability before printing the tickets.

After a ticket sale, it must update the number of allocated seats to prevent over-booking.

The theater includes three sections:

* the front section, which has 50 seats costing $30 each
* the center section, which has 100 seats costing $20 each, and
* the back section, which has 50 seats costing $10 each

You start by considering the problem statement. There are two inputs you'll need to use – the number of seats requested and where they are located.

Graphic

*The Input column of the table has two entries – no\_of\_seats and theater\_section. The rest of the table is still empty.*

Next you consider the required processes. The program needs to retrieve the number of requested seats and the theater section from the customer's input. Then it needs to match the customer's section to the code that represents it. It also has to check for seat availability and calculate the total price of a ticket.

Graphic

*The Process column of the table now contains the following pseudocode:  
GET no\_of\_seats  
GET theater\_section  
SELECT code for relevant theater\_section  
CHECK available\_seats  
CALCULATE price  
PRINT tickets*

Last the program has to print the ticket, update the number of seats allocated for that section, and calculate and display the total cost of the ticket to the customer.

Graphic

*Two additional processes have been added to the Process column – CALCULATE seats\_allocated and DISPLAY total\_price.*

There are two outputs for this problem. The program must display the total cost of the ticket to the customer and print the ticket. Both outputs involve a process and should be indicated in the table as such.

Graphic

*Two entries have been added to the Output column – total\_price and tickets.*

Once you've finished creating a defining table, you can use it to guide you through the rest of the process.

You can now start preparing the pseudocode algorithm for the program. The program must branch to the code that deals with each of the theater sections – front, center, and back – depending on which one the customer wants.

The program must also check that enough seats are available in that section, calculate the ticket price, and update the number of seats allocated.

You start the algorithm by creating statements that represent the processing needed to capture the input data.

There are three sections in the theater, so you use a CASE statement to branch to the relevant code for each theater section.

Graphic

*The relevant code is  
  
CASE theater\_section IS*

Code

GET no\_of\_seats  
GET theater\_section  
CASE theater\_section IS  
   front :  
  center :  
    back :  
ENDCASE

The IF statement first checks how many seats are currently available by subtracting the number of seats already allocated from the maximum capacity of the section, which is 50 in this case. The IFstatement then compares the difference to the number of seats requested.

Graphic

*The relevant code is  
  
IF no\_of\_seats <= (50-front\_allocated) THEN*

Code

GET no\_of\_seats  
GET theater\_section  
CASE theater\_section IS  
   front : IF no\_of\_seats <= (50 - front\_allocated) THEN  
             ticket\_price = no\_of\_seats \* 30  
             PRINT ticket  
             front\_allocated = front\_allocated + no\_of\_seats  
           ELSE  
             OUTPUT "Not enough seats left in this section"  
           ENDIF  
  center :  
    back :  
ENDCASE

The program continues processing a ticket request only if tickets in the relevant section are available. It then calculates the ticket price and prints the ticket.

Graphic

*The relevant code is  
  
ticket\_price = no\_of\_seats \* 30  
PRINT ticket*

Code

GET no\_of\_seats  
GET theater\_section  
CASE theater\_section IS  
   front : IF no\_of\_seats <= (50 - front\_allocated) THEN  
             ticket\_price = no\_of\_seats \* 30  
             PRINT ticket  
             front\_allocated = front\_allocated + no\_of\_seats  
           ELSE  
             OUTPUT "Not enough seats left in this section"  
           ENDIF  
  center :  
    back :  
ENDCASE

Once the tickets have been printed, the number of seats sold is added to the total number of seats already allocated in the section. This ensures that the number of seats listed as available is kept up to date.

Graphic

*The relevant code is  
  
front\_allocated = front\_allocated + no\_of\_seats*

Code

GET no\_of\_seats  
GET theater\_section  
CASE theater\_section IS  
   front : IF no\_of\_seats <= (50 - front\_allocated) THEN  
             ticket\_price = no\_of\_seats \* 30  
             PRINT ticket  
             front\_allocated = front\_allocated + no\_of\_seats  
           ELSE  
             OUTPUT "Not enough seats left in this section"  
           ENDIF  
  center :  
    back :  
ENDCASE

If the number of seats requested exceeds those available, the ELSE statement is triggered and the program prints a message stating that there are not enough seats left in the section.

Graphic

*The relevant code is  
  
ELSE OUTPUT "Not enough seats left in this section"*

Code

GET no\_of\_seats  
GET theater\_section  
CASE theater\_section IS  
   front : IF no\_of\_seats <= (50 - front\_allocated) THEN  
             ticket\_price = no\_of\_seats \* 30  
             PRINT ticket  
             front\_allocated = front\_allocated + no\_of\_seats  
           ELSE  
             OUTPUT "Not enough seats left in this section"  
           ENDIF  
  center :  
    back :  
ENDCASE

You can adjust the code for the front theater section to reuse it for the center and back sections. You simply replace the number of seats and the ticket price per section in each case.

Code

GET no\_of\_seats  
GET theater\_section  
CASE theater\_section IS  
   front : IF no\_of\_seats <= (50 - front\_allocated) THEN  
             ticket\_price = no\_of\_seats \* 30  
             PRINT ticket  
             front\_allocated = front\_allocated + no\_of\_seats  
           ELSE   
             OUTPUT "Not enough seats left in this section"  
           ENDIF  
  center : IF no\_of\_seats <= (100 - front\_allocated) THEN  
             ticket\_price = no\_of\_seats \* 20  
             PRINT ticket  
             front\_allocated = front\_allocated + no\_of\_seats  
           ELSE   
             OUTPUT "Not enough seats left in this section"  
           ENDIF  
    back : IF no\_of\_seats <= (50 - front\_allocated) THEN  
             ticket\_price = no\_of\_seats \* 10  
             PRINT ticket  
             front\_allocated = front\_allocated + no\_of\_seats  
           ELSE   
             OUTPUT "Not enough seats left in this section"  
           ENDIF  
ENDCASE

You can now write the algorithm pseudocode in a programming language – C in this case.

You can convert the pseudocode to a programming language one section at a time.

Code

no\_of\_seats = scanf ("%d",&no\_of\_seats);  
theater\_section = scanf ("%c",&theater\_section);  
switch (theater\_section){  
case 'F' :   
    if (no\_of\_seats <= (50 -  front\_allocated))  
    {ticket\_price = no\_of\_seats \* 30;  
    printf("Front seats cost $%d", ticket\_price, "\n");  
    front\_allocated = front\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;  
case 'C' :   
    if (no\_of\_seats <= (100 -  center\_allocated))  
    {ticket\_price = no\_of\_seats \* 20;  
    printf("Center seats cost $%d", ticket\_price, "\n");  
    center\_allocated = center\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;   
case 'B' :  
    if (no\_of\_seats <= (50 -  back\_allocated))  
    {ticket\_price = no\_of\_seats \* 10;  
    printf("Back seats cost $%d", ticket\_price, "\n");  
    back\_allocated = back\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;}

**GET no\_of\_seats  
GET theater\_section**

To input the number of seats and the theater section requested by the user, you use the scanf statement, which reads information from an input device like a keyboard. Because the number of seats is an integer, you use the %d option in the scanf statement. This accepts an integer as input and then writes this to the variable no\_of\_seats. The value of the variable theater\_section will be a character, so you use the %c option to read it.   
  
**Code**  
no\_of\_seats = scanf ("%d",&no\_of\_seats);  
theater\_section = scanf ("%c",&theater\_section);  
switch (theater\_section){  
case 'F' :   
    if (no\_of\_seats <= (50 -  front\_allocated))  
    {ticket\_price = no\_of\_seats \* 30;  
    printf("Front seats cost $%d", ticket\_price, "\n");  
    front\_allocated = front\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;  
case 'C' :   
    if (no\_of\_seats <= (100 -  center\_allocated))  
    {ticket\_price = no\_of\_seats \* 20;  
    printf("Center seats cost $%d", ticket\_price, "\n");  
    center\_allocated = center\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;   
case 'B' :  
    if (no\_of\_seats <= (50 -  back\_allocated))  
    {ticket\_price = no\_of\_seats \* 10;  
    printf("Back seats cost $%d", ticket\_price, "\n");  
    back\_allocated = back\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;}

**CASE theater\_section IS  
front:**

The switch statement is used to perform conditional logic with case statements in C. After the case statement, the theater\_section variable follows in parentheses to indicate that the value of this variable determines the code that executes next. The theater\_section variable can equal one of the three values – F, C, or B.   
  
**Code**  
no\_of\_seats = scanf ("%d",&no\_of\_seats);  
theater\_section = scanf ("%c",&theater\_section);  
switch (theater\_section){  
case 'F' :   
    if (no\_of\_seats <= (50 -  front\_allocated))  
    {ticket\_price = no\_of\_seats \* 30;  
    printf("Front seats cost $%d", ticket\_price, "\n");  
    front\_allocated = front\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;  
case 'C' :   
    if (no\_of\_seats <= (100 -  center\_allocated))  
    {ticket\_price = no\_of\_seats \* 20;  
    printf("Center seats cost $%d", ticket\_price, "\n");  
    center\_allocated = center\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;   
case 'B' :  
    if (no\_of\_seats <= (50 -  back\_allocated))  
    {ticket\_price = no\_of\_seats \* 10;  
    printf("Back seats cost $%d", ticket\_price, "\n");  
    back\_allocated = back\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;}

**IF no\_of\_seats <= (50-front\_allocated) THEN**

Before processing the ticket request, the program checks that there are enough seats left in the selected section. If the condition is true it enters the if code block.   
  
**Code**  
no\_of\_seats = scanf ("%d",&no\_of\_seats);  
theater\_section = scanf ("%c",&theater\_section);  
switch (theater\_section){  
case 'F' :   
    if (no\_of\_seats <= (50 -  front\_allocated))  
    {ticket\_price = no\_of\_seats \* 30;  
    printf("Front seats cost $%d", ticket\_price, "\n");  
    front\_allocated = front\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;  
case 'C' :   
    if (no\_of\_seats <= (100 -  center\_allocated))  
    {ticket\_price = no\_of\_seats \* 20;  
    printf("Center seats cost $%d", ticket\_price, "\n");  
    center\_allocated = center\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;   
case 'B' :  
    if (no\_of\_seats <= (50 -  back\_allocated))  
    {ticket\_price = no\_of\_seats \* 10;  
    printf("Back seats cost $%d", ticket\_price, "\n");  
    back\_allocated = back\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;}

**ticket\_price = no\_of\_seats \* 30  
PRINT ticket  
front\_allocated = front\_allocated - no\_of\_seats**

Next the program calculates the total price of the tickets and prints a message to the output device to show this total. The code inside the printf statement, /n, is the new line character, which moves the cursor to the next line on the output device. Lastly, it adjusts the number of the seats already allocated in the selected section.   
  
**Code**  
no\_of\_seats = scanf ("%d",&no\_of\_seats);  
theater\_section = scanf ("%c",&theater\_section);  
switch (theater\_section){  
case 'F' :   
    if (no\_of\_seats <= (50 -  front\_allocated))  
    {ticket\_price = no\_of\_seats \* 30;  
    printf("Front seats cost $%d", ticket\_price, "\n");  
    front\_allocated = front\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;  
case 'C' :   
    if (no\_of\_seats <= (100 -  center\_allocated))  
    {ticket\_price = no\_of\_seats \* 20;  
    printf("Center seats cost $%d", ticket\_price, "\n");  
    center\_allocated = center\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;   
case 'B' :  
    if (no\_of\_seats <= (50 -  back\_allocated))  
    {ticket\_price = no\_of\_seats \* 10;  
    printf("Back seats cost $%d", ticket\_price, "\n");  
    back\_allocated = back\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;}

**ELSE OUTPUT "Not enough seats left in this section"**

When there are not enough seats left, the if condition is false. The program then prints an error message to the output device.   
  
**Code**  
no\_of\_seats = scanf ("%d",&no\_of\_seats);  
theater\_section = scanf ("%c",&theater\_section);  
switch (theater\_section){  
case 'F' :   
    if (no\_of\_seats <= (50 -  front\_allocated))  
    {ticket\_price = no\_of\_seats \* 30;  
    printf("Front seats cost $%d", ticket\_price, "\n");  
    front\_allocated = front\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;  
case 'C' :   
    if (no\_of\_seats <= (100 -  center\_allocated))  
    {ticket\_price = no\_of\_seats \* 20;  
    printf("Center seats cost $%d", ticket\_price, "\n");  
    center\_allocated = center\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;   
case 'B' :  
    if (no\_of\_seats <= (50 -  back\_allocated))  
    {ticket\_price = no\_of\_seats \* 10;  
    printf("Back seats cost $%d", ticket\_price, "\n");  
    back\_allocated = back\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;}

**ENDCASE**

The break statement causes the program to exit the switch statement. If it is omitted, the program will continue to evaluate the remaining case conditions.   
  
**Code**  
no\_of\_seats = scanf ("%d",&no\_of\_seats);  
theater\_section = scanf ("%c",&theater\_section);  
switch (theater\_section){  
case 'F' :   
    if (no\_of\_seats <= (50 -  front\_allocated))  
    {ticket\_price = no\_of\_seats \* 30;  
    printf("Front seats cost $%d", ticket\_price, "\n");  
    front\_allocated = front\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;  
case 'C' :   
    if (no\_of\_seats <= (100 -  center\_allocated))  
    {ticket\_price = no\_of\_seats \* 20;  
    printf("Center seats cost $%d", ticket\_price, "\n");  
    center\_allocated = center\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;   
case 'B' :  
    if (no\_of\_seats <= (50 -  back\_allocated))  
    {ticket\_price = no\_of\_seats \* 10;  
    printf("Back seats cost $%d", ticket\_price, "\n");  
    back\_allocated = back\_allocated - no\_of\_seats;}  
    else printf("Not enough seats left.\n");  
break;}

**Question Set**

You've now been given an algorithm in pseudocode that keeps track of customers' purchases.   
  
Each customer is categorized as belonging to "Bronze", "Silver", or "Gold" categories depending on the value of their purchases. Silver and Gold customers qualify for discounted purchases.   
  
The algorithm can be implemented in any programming language, but you decide to use C.

**Question 1 of 3**

Question

To start, you want to read the value of the customer's total purchases, which is an integer value, from the keyboard.  
  
Complete the code to do this.

**Code**  
totalPurchases = INSERT THE MISSING CODE("%d",&totalPurchases);   
...

Answer

*The scanf method reads data from the keyboard.*

**Correct answer(s):**

1. scanf

**Question 2 of 3**

Question

Next you want to find out if the customer qualifies as a Bronze, Silver, or Gold customer. These levels are represented by the B, S, and G characters. You read this information from the keyboard using the scanf method.  
  
Complete the code that instructs scanf to receive data in the form of characters.

**Code**  
totalPurchases = scanf ("%d",&totalPurchases);   
preferredDiscount = scanf (INSERT THE MISSING CODE,&preferredDiscount);

Answer

*The "%c" parameter indicates to the scanf method that a character is expected from the keyboard.*

**Correct answer(s):**

1. "%c"

**Question 3 of 3**

Question

You have written the CASE block for the Bronze customer level.   
  
Complete the code to ensure the program doesn't execute the Silver and Gold code after running the Bronze processes.

**Code**  
totalPurchases = scanf ("%d",&totalPurchases);   
preferredDiscount = scanf ("%c",&preferredDiscount);  
switch (preferredDiscount)  
{  
  case 'B' : if (totalPurchases < 1000)  
                  {  
                      discount = 0;   
                      printf("Sorry, no discount. Spend $1000 or more to receive a discount.\n");  
                  }  
                  else  
                  {  
                      printf("Welcome back, preferred customer!\n");  
                  }  
                  INSERT THE MISSING CODE;

Answer

*You use the break command to exit the switch statement once a case statement has been executed.*

**Correct answer(s):**

1. break

**3. Summary**

You can follow four steps to simplify the process of solving a programming problem. You analyze the problem statement, create a table defining the required inputs, processes, and outputs, use the table to guide you in writing a pseudocode algorithm, and finally convert the pseudocode into a programming language.  
  
If program inputs are limited to a number of specific possibilities, you can use branching statements to create the program. You can use CASE statements to decide on the number of actions to take. You can also nest IF and ELSE block statements to further branch the processes, as required.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a03_it_enus/output/html/course_transcript.html#top)

Completing Selection Statements

Learning Objectives

*After completing this topic, you should be able to*

* *complete IF statements in a given example*
* *complete CASE statements in a given example*
* *complete a program based on a defining table in a given example*

**1. Exercise overview**

In this exercise, you're required to recognize how to use selection statements in programming.

This involves the following tasks:

* completing IF statements
* completing CASE statements, and
* using a defining table to complete an algorithm

**2. Completing IF statements**

Question

You are coding registration software for a university. The university is offering $1,000 discounts to the first 100 students to register.  
  
Complete the IF statement so that the discount is correctly applied.

**Code**  
IF student\_no INSERT THE MISSING CODE THEN  
Discount = 1000

**Options:**

1. <= 100
2. >= 100
3. < 100

Answer

***Option 1:****Correct. The code <=100 will ensure that the first 100 students get the $1,000 discount.*

***Option 2:****Incorrect. The code >=100 would cause the students with the $1,000 discount to overlap with the students getting the $500 discount.*

***Option 3:****Incorrect. The code <100 would cause the hundredth student to register to be excluded from the discounts.*

**Correct answer(s):**

1. <= 100

Question

Another $500 discount is offered to the next 100 students to register.  
  
Complete the ELSE IF statement to ensure that the students from 101 to 200 are given the $500 discount.

**Code**  
IF student\_no <= 100 THEN  
    Discount = 1000  
ELSE   
    IF student\_no > 100 AND student\_no INSERT THE MISSING CODE THEN  
        Discount = 500  
    ELSE  
        Discount = 0  
    END IF  
END IF

Answer

*Correct. The code <= 200 ensures that the students from 101 to 200 are given the $500 discount.*

**Correct answer(s):**

1. <= 200

**3. Completing CASE statements**

Question

There are a variety of student discounts, which are classified according to category. The student categories are P for part-time, F for full-time, and X for extra-mural.   
  
Which discount does a part-time student receive?

**Code**  
CASE student category of :  
  
X: discount = 1500  
  
P: discount = 3750  
  
F: discount = 6500  
  
other: discount = 0   
  
ENDCASE

**Options:**

1. $3,750
2. $1,500
3. $6,500

Answer

***Option 1:****Correct. If a student is part-time, they fall into category P. The discount for category P is $3,750.*

***Option 2:****Incorrect. The discount for category X is $1,500. This discount is applied to extra-mural students. If a student is part-time, they fall into category P, which has a discount of $3,750.*

***Option 3:****Incorrect. The discount for category F is $6,500. This discount is applied to full-time students. If a student is part-time, they fall into category P, which has a discount of $3,750.*

**Correct answer(s):**

1. $3,750

**4. Using a defining table**

You want to write the section of the program that calculates the total course fees and prints a receipt that specifies the discounted price.

Question

You have created the defining table and have already added the inputs and outputs. Identify the first process that will be added to the Process column.

*A defining table contains course\_fee and student\_category in the Input column, and total\_ price and receipt in the output column. The processes column is blank.*

**Options:**

1. SELECT category\_discount
2. PRINT receipt
3. CALCULATE total\_price
4. GET course\_fee, student\_category

Answer

***Option 1:****Incorrect. SELECT category\_discount will be performed after retrieving the course\_feeand student\_category values.*

***Option 2:****Incorrect. PRINT receipt will be the final process in the table.*

***Option 3:****Incorrect. CALCULATE total\_price will be performed after the course\_fee, student\_category, and category\_discount values are created.*

***Option 4:****Correct. GET course\_fee, student\_category will be the first process in the defining table to retrieve data inputs from the user.*

**Correct answer(s):**

4. GET course\_fee, student\_category

Question

Identify the first process that will use input to produce the desired outputs.

*GET course\_fee, student\_category  
has been added to the first row in the Process column.*

**Options:**

1. CALCULATE total\_price
2. PRINT receipt
3. SELECT category\_discount

Answer

***Option 1:****Correct. CALCULATE total\_price will use the two user inputs with the category\_discount value to calculate the total price.*

***Option 2:****Incorrect. PRINT receipt is the final process in the defining table and will print all necessary information to the output device.*

***Option 3:****Incorrect. SELECT category\_discount is the second process in the defining table and forms part of the calculation that outputs total\_price.*

**Correct answer(s):**

1. CALCULATE total\_price

Question

The student categories are P for part-time ($3,500), F for full-time ($5,000), and X for extra mural ($1,250), which form part of a CASE statement.   
  
Match each category with the algorithm used to calculate the total price.

**Options:**

1. X:
2. P:
3. F:
4. Other:

**Targets:**

1. total\_price = course\_fee - 1250
2. total\_price = course\_fee - 0
3. total\_price = course\_fee - 5000
4. total\_price = course\_fee - 3500

Answer

*Extra-mural students get a discount of $1,250.*

*Students who don't fall into these categories get no discount on their course fees.*

*Full-time students get a discount of $5,000.*

*Part-time students get a discount of $3,500.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option D

Target 3 = Option C

Target 4 = Option B

IF and CASE statements have been completed, and a defining table has been used to complete an algorithm.

Code

GET discount   
  
CASE student category OF  
  
X:  total\_price = course\_fee – 1250  
     PRINT receipt   
P:  total\_price = course\_fee – 3500  
     PRINT receipt   
F:  total\_price = course\_fee – 5000  
     PRINT receipt   
other:  total\_price = course\_fee – 0  
     PRINT receipt   
  
ENDCASE

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a03_it_enus/output/html/course_transcript.html#top)

FOR Loops

Learning Objectives

*After completing this topic, you should be able to*

* *sequence the operations in a FOR loop*
* *recognize how to use the FOR loop in computer programs*

**1. Introducing FOR loops**

You may need to execute either single or compound statements a number of times in a program. If you want to create a piece of code to print five address labels, you could write a print statement and repeat it five times, for example.

Code

GET name\_and\_address  
    PRINT name\_and\_address  
    PRINT name\_and\_address  
    PRINT name\_and\_address  
    PRINT name\_and\_address  
    PRINT name\_and\_address

This is a rather inefficient method, especially if you are printing a large number of labels. The amount of duplicated code is time-consuming to write, and difficult for other programmers to read. Also, overly large programs use up memory and processing resources. The leaner a program is, the better its performance.

Code

GET name\_and\_address  
    PRINT name\_and\_address  
    PRINT name\_and\_address  
    PRINT name\_and\_address  
    PRINT name\_and\_address  
    PRINT name\_and\_address

To avoid creating unwieldy code, you can use a programming procedure called a *loop*to avoid using the same statement repeatedly.

Graphic

*In this diagram, the program is told to print a statement 5 times. At the beginning of the FOR loop, the variable count equals 0. Each time a statement is printed the program adds 1 to count and checks to see if count is greater than 4 (i.e., has looped 5 times). If it hasn’t then the program loops again, printing another statement. Once 5 iterations have been processed, count will equal 5 and the program will end the loop.*

A FOR loop lets you run a task a predetermined number of times. It removes the need to repeat code, making the code shorter.

Conditions in a program can be used to determine a variable for the number of times the task has to be executed.

A FOR loop can contain only one command or encompass an entire program that needs to be run several times before terminating. Each time the code in a FOR loop runs, the cycle is known as an *iteration*.

A FOR loop performs specific steps in sequence:

**the counter is set to an initial value**

The *counter value* is set before code in the FOR loop is executed. You set the increment or decrement value, which is usually one.

**if the counter value hasn't reached its limit, the instructions included in the loop are executed**

The loop will continue to repeat as long as the counter value remains inside the range set by the loop limit.

**the counter is incremented or decremented, and**

Because the counter is used to control the number of times the loop is executed, it's incremented or decremented after the instructions included in the loop are executed.

**the loop is terminated when the counter reaches its limit**

The loop stops executing as soon as the counter exceeds the limit.

You can write a program so that when it executes, it asks the user to set the number of times a FORloop must run. The user can be prompted to specify a number as the limit of the counter value – for example, to set how many copies of an invoice to print.

Code

INPUT "How many copies of the Invoice do you need printed? "; number$  
FOR count = 1 to number$       
    'the code to print the invoice details  
NEXT count

Question

Identify the order of the steps a program takes when it executes a FOR loop.

**Options:**

1. Initialize the counter in the loop to an initial value
2. If the counter hasn't reached its limit, execute the instructions included in the FOR loop
3. Increment or decrement the counter
4. Terminate the loop when the counter reaches its limit

Answer

**Correct answer(s):**

**Initialize the counter in the loop to an initial value is ranked**

Before the code in a FOR loop is executed, the counter in the loop is initialized, with the increment or decrement value usually set to one.

**If the counter hasn't reached its limit, execute the instructions included in the FORloop is ranked**

The loop will continue to repeat as long as the counter remains inside the range set by the counter limit.

**Increment or decrement the counter is ranked**

Because the counter is used to control the number of times the loop is executed, it's incremented or decremented after the instructions included in the loop are executed.

**Terminate the loop when the counter reaches its limit is ranked**

Once the counter exceeds the specified value, the loop will stop running.

**2. Using FOR loops**

In a FOR loop to print a specific number of address labels, the first line of code receives the required input details. Then the address details are specified, and finally the labels are printed.

Code

Name$ = "Joe Bloggs"  
Address$ = "12 Basic Language Avenue"  
FOR count = 1 to 5  
    PRINT "Name: "; Name$; " Address: "; Address$  
NEXT count

You don't have to repeat the code for each statement because the single FOR loop can be set to make the program print five labels.

Code

Name$ = "Joe Bloggs"  
Address$ = "12 Basic Language Avenue"  
FOR count = 1 to 5  
    PRINT "Name: "; Name$; " Address: "; Address$  
NEXT count

The loop is created by using a counter variable called count, with a starting value of one. The number of iterations is set to five. The count is incremented by one every time the loop is executed.

Code

Name$ = "Joe Bloggs"  
Address$ = "12 Basic Language Avenue"  
FOR count = 1 to 5  
    PRINT "Name: "; Name$; " Address: "; Address$  
NEXT count

The counter is used to terminate the loop once the specified number has been reached. So once the count exceeds five, the loop will terminate.

Code

Name$ = "Joe Bloggs"  
Address$ = "12 Basic Language Avenue"  
FOR count = 1 to 5  
    PRINT "Name: "; Name$; " Address: "; Address$  
NEXT count

You want to write a program to calculate the total mass of the cargo in a shipping container, which varies according to the number of parcels in the container.

You first need to retrieve the input details. So you list the number of packages and the package mass in each container as input elements in a defining table.

Graphic

*A table contains Input, Process, and Output columns. The Input column contains no\_of\_packages and package\_mass.*

Processes that must run to generate a result include getting the number of packages in each container, calculating the individual mass of the packages, and calculating the sum of their mass.

Graphic

*The Process column contains GET no\_of\_packages, CALCULATE package\_mass, and CALCULATE total\_package\_mass.*

The total package mass is the output value.

Graphic

*The Output column contains total\_package\_mass.*

You begin this pseudocode algorithm with a request for the number of packages in the container.

Code

PRINT 'How many packages are in the container?'  
    GET no\_of\_packages

You can use a FOR loop to get the mass of each package and add it to the mass total.

Code

FOR i = 1 to no\_of\_packages  
    GET package\_mass  
    total\_package\_mass = total\_package\_mass + package\_mass  
ENDFOR

The number entered by the user determines how many times the loop will execute.

Code

GET no\_of\_packages  
FOR i = 1 to no\_of\_packages  
    GET package\_mass  
    total\_package\_mass = total\_package\_mass + package\_mass  
ENDFOR  
PRINT total\_package\_mass

In most languages, the default loop control variable increment step is 1.

You can change this by adding the STEP statement and specifying the step size as any number.

Code

FOR count = 1 TO 10 STEP 2  
    PRINT count  
NEXT count

When executed, the result displays every second number from one to nine.

Graphic

*The output in the example is 1 3 5 7 9.*

You can also decrement the control variable by replacing to with downto.

Code

FOR count = 10 DOWNTO 1   
    PRINT count  
NEXT count

When executed, the code produces the numbers ten down to one.

Graphic

*The output is 10 9 8 7 6 5 4 3 2 1.*

Question

You're creating a FOR loop to print labels containing clients' contact details. The variable numberOfClients will be used to determine how many labels to print.  
  
Identify the pseudocode that will enable the loop to run for the required number of iterations.

**Code**  
GET numberOfClients  
    FOR printClientInfo = INSERT THE MISSING CODE  
        GET clientName  
        GET clientPhoneNumber  
        GET clientAddress  
        PRINT clientName  
        PRINT clientPhoneNumber  
        PRINT clientAddress  
    ENDFOR

**Options:**

1. 1 to numberOfClients
2. GET printClientInfo
3. NEW printClientInfo

Answer

***Option 1:****Correct. You use 1 to numberOfClients to begin the loop and specify the number of times that the FOR loop will execute.*

***Option 2:****Incorrect. The correct way to begin a loop is to provide the loop count variable range.*

***Option 3:****Incorrect. The correct way to begin a loop is to provide the loop count variable range.*

**Correct answer(s):**

1. 1 to numberOfClients

**3. Nesting FOR loops**

A loop can run within another loop. Loops that appear inside other loops are known as *nested loops*.

Graphic

*The diagram has a Nested loop contained in an Outer loop. The Outer loop code is: FOR X = 1 TO 3 PRINT "Outer loop run #"; X and NEXT X. The Nested loop code is FOR Y = 1 TO 5 PRINT " Nested loop run #"; Y and NEXT Y.*

Every time the outside loop runs through an iteration, the inside loop is processed, running all its iterations.

For each iteration of the outside loop, the inner loop goes through all its iterations. So nested loops process more iterations than outer loops.

It can be very difficult to follow the code for a program if you include a large number of nested loops within one loop. So as a general rule, nest only one loop inside another.

This pseudocode example is designed to generate reports for three students, each with five different test grades that are stored in the grades array. The program gets a student's name once and then processes that student's five grades.

Code

FOR X = 1 to 3  
    GET studentName from the array  
    PRINT "Student " studentName " grades"  
    FOR Y = 1 to 5  
        GET gradeAmount from the array  
        PRINT "Grade #" Y " for student " studentName " is " gradeAmount  
    NEXT Y  
NEXT X

As defined by the pseudocode, each time the outer loop is run, the inner loop runs five times. Ultimately, the nested loop will run 15 times, whereas the outer loop will run only three times – once per student.

Code

FOR X = 1 to 3  
    GET studentName from the array  
    PRINT "Student " studentName " grades"  
    FOR Y = 1 to 5  
        GET gradeAmount from the array  
        PRINT "Grade #" Y " for student " studentName " is " gradeAmount  
    NEXT Y  
NEXT X

The final output includes students' names and their grades.

Code

Student John Sands grades  
Grade #1 for student John Sands is 86  
Grade #2 for student John Sands is 62  
Grade #3 for student John Sands is 90  
Grade #4 for student John Sands is 75  
Grade #5 for student John Sands is 78  
  
Student Amy Smith grades  
Grade #1 for student Amy Smith is 91  
Grade #2 for student Amy Smith is 88  
Grade #3 for student Amy Smith is 91  
Grade #4 for student Amy Smith is 82  
Grade #5 for student Amy Smith is 78  
  
Student Elaine Evans grades  
Grade #1 for student Elaine Evans is 77  
Grade #2 for student Elaine Evans is 79  
Grade #3 for student Elaine Evans is 83  
Grade #4 for student Elaine Evans is 70  
Grade #5 for student Elaine Evans is 84

**4. Summary**

You can use FOR loops to execute statements a set number of times, for a predetermined number of cycles.   
  
Each time a FOR loop runs, the commands it contains execute and a counter is incremented or decremented by a set value – usually one, although you can change this. Once the value of the counter reaches a specified limit, the loop terminates.  
  
Loops that appear inside other loops are called nested or inner loops. Each time an outer loop runs, the inner loop iterates a specified number of times.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a03_it_enus/output/html/course_transcript.html#top)

WHILE Loops

Learning Objective

*After completing this topic, you should be able to*

* *recognize how to use the WHILE loop in computer programs*

**1. Using WHILE loops**

When you write a computer program, it may be necessary to test a single condition multiple times. However, it may not always be feasible to add lines of code for every instance because you may not know in advance how many times the condition will be tested.

For example, you may want to process numbers from a list of unspecified length. Because you don't know how many numbers there are to be processed, you don't know how many times the loop should be executed. You can use a WHILE loop to resolve this problem.

The WHILE statement is an entry-condition or loop pre-test. This means that a program won't execute the statement block within the loop unless the specified condition is met.

In this pseudocode example, the condition is that the number entered in the algorithm must be less than 3.

Code

NUM = 2  
WHILE NUM < 3  
    PRINT NUM  
    NUM = NUM + 1   
ENDWHILE

If the condition is immediately false, as it is in this example, the loop won't execute at all.

Code

NUM = 7  
WHILE NUM < 3  
    PRINT NUM  
    NUM = NUM + 1   
ENDWHILE

A WHILE loop runs indefinitely unless the loop condition becomes false. In this case, the value of NUM will never change – so the WHILE loop will never stop iterating.

Code

NUM = 1  
WHILE NUM < 3  
    PRINT NUM  
ENDWHILE  
PRINT "The final number is", NUM

To ensure that this doesn't happen, a WHILE loop should include a statement that changes the value of the conditional expression. In this example, the value of NUM is incremented by 1, so at some point in the loop's execution, it should fail the tested condition.

Graphic

*The relevant code is:  
  
NUM = NUM + 1*

Code

NUM = 1  
WHILE NUM < 3  
    PRINT NUM  
    NUM = NUM + 1   
ENDWHILE  
PRINT "The final number is", NUM

To determine whether the loop should be executed again, the loop condition is queried after each iteration. When the loop condition becomes false, the loop terminates and the program moves on to the statements following the loop. In this example, the value of NUM will be printed until its value is greater than 2. After processing two iterations, NUM will equal 3 and the WHILE loop will terminate.

Graphic

*The relevant code is:  
  
PRINT "The final number is", NUM*

Code

NUM = 1  
WHILE NUM < 3  
    PRINT NUM  
    NUM = NUM + 1   
ENDWHILE  
PRINT "The final number is", NUM

To write an algorithm that uses a WHILE loop to print several labels, you first complete a defining table for the problem statement.

Graphic

*The table has three columns: INPUTS, PROCESSES, and OUTPUTS. In the INPUTS column there is one row with the value answer.  
  
In the PROCESSES column, there are four rows: PRINT name and address, PRINT " Do you wish to print another label?", GET answer, and PRINT "Thank you".  
  
In the OUTPUTS column, there are two rows: name and address, and End message.*

You then create this pseudocode algorithm to capture the user's details, print a label, and ask whether another label is required.

Code

answer = "yes"  
    WHILE answer = "yes"  
        PRINT name and address      
        PRINT "Do you wish to print another label?"       
        GET answer   
    ENDWHILE   
PRINT "Thank you"

The loop condition is answer = "yes". As long as this condition is true, the loop will continue to execute.

Code

answer = "yes"  
    WHILE answer = "yes"  
        PRINT name and address      
        PRINT "Do you wish to print another label?"       
        GET answer   
    ENDWHILE   
PRINT "Thank you"

Because the variable answer is set to yes as the loop begins its execution, the loop will execute at least once.

Graphic

*The relevant code is  
  
answer = "yes"  
   WHILE answer = "yes"*

Code

answer = "yes"  
    WHILE answer = "yes"  
        PRINT name and address      
        PRINT "Do you wish to print another label?"       
        GET answer   
    ENDWHILE   
PRINT "Thank you"

After the first label has been printed, the user will have the option to end the process by entering a response other than yes, when prompted about printing more labels.

Graphic

*The relevant code is  
  
PRINT "Do you wish to print another label?"       
       GET answer*

Code

answer = "yes"  
    WHILE answer = "yes"  
        PRINT name and address      
        PRINT "Do you wish to print another label?"  
        GET answer   
    ENDWHILE   
PRINT "Thank you"

The user response is then evaluated against the loop condition. If the answer is yes, the loop will iterate again.  
  
Because the answer is no in this case, the loop won't execute again. Instead the next line of the program will execute.

Graphic

*The relevant pieces of code are  
  
WHILE answer = "yes"  
and  
PRINT "Thank you"*

Code

answer = "yes"  
    WHILE answer = "yes"  
        PRINT name and address      
        PRINT "Do you wish to print another label?"       
        GET answer   
    ENDWHILE   
PRINT "Thank you"

Question

In this pseudocode example, how many times will this WHILE loop execute?

**Code**  
num = 7  
WHILE num < 3  
    num=num+1  
    PRINT num   
ENDWHILE  
PRINT num

**Options:**

1. Never
2. After a single iteration
3. After seven iterations
4. After three iterations

Answer

***Option 1:****Correct. The condition is immediately false so the loop won't execute at all. Because the WHILE loop is an entry condition loop, a condition must be met or the loop won't execute.*

***Option 2:****Incorrect. The loop won't execute at all as the condition will be immediately false.*

***Option 3:****Incorrect. The loop won't execute at all because the initial tested condition will evaluate to false.*

***Option 4:****Incorrect. The loop will never be executed because the initial tested condition is false.*

**Correct answer(s):**

1. Never

Depending on the programming language you're using, the syntax of the WHILE loop may be slightly different.

In this example, BASIC has been used to create a program to print labels. The loop ensures that labels will continue to print until the user enters 'N' – or anything other than 'Y'.

Graphic

*The relevant code is:   
  
WHILE answer$ = 'Y'*

Code

answer$ = "Y"  
WHILE answer$ = "Y"  
    PRINT name$  
    PRINT address$  
    PRINT "Do you want to print another? Y/N"  
    INPUT answer$  
WEND

In this example, the program to print labels is written in Pascal. Unlike in the BASIC example, the WHILE statement includes a condition followed by the keyword DO.   
  
Also, the statements that make up the loop are placed between BEGIN and END tags.

Code

while answer = "Y" do  
begin          
    writeln (name);          
    writeln (address);          
    writeln ("Do you want to print another ? Y/N");  
    read (answer)  
end;

This is the same program written in C. In the tested condition, the relational operator "==" means equal to.

Graphic

*The relevant code is:  
  
(answer == 'Y')*

Code

while (answer == 'Y')  
{        
    printf ("%s\n" , name);        
    printf ("%s\n" , address);       
    printf ('Do you want to print another? Y/N');       
    scanf ("%s\n" , answer);   
}

The statement is followed by the looped print and scanf statements, which are placed between opening and closing curly braces.

Code

while (answer == 'Y')  
{        
    printf ("%s\n" , name);        
    printf ("%s\n" , address);       
    printf ('Do you want to print another? Y/N');       
    scanf ("%s\n" , answer);   
}

You can nest WHILE loops in the same way that you nest FOR loops.

Graphic

*The algorithm starts and tests choice 1. If it is false then the loop ends. If it is true, then a second loop condition (choice 2) is tested. If choice 2 is false he loop ends. If it is true, the code will execute and the condition will be tested again. When the second loop condition is false, it will test choice 1 again.*

However, it's generally best to avoid doing this because it increases the risk that you'll include a WHILE loop that will continue executing indefinitely and so cause a program to hang or crash.

It also isn't a good idea to nest too many loops because this can make the structure of a program very complex.

You can also nest WHILE loops inside FOR loops, or nest FOR loops inside of WHILE loops.  
  
This is an example of a BASIC program that uses a WHILE loop nested inside a FOR loop. It's a counting program designed to output five grades for each of three students in a database.

Code

FOR X = 1 to 3  
    'write code to get the studentName from the array  
    PRINT "Student "; studentName " grades"  
    Y = 1  
    WHILE Y <= 5  
         'write code to get the gradeAmount from the array  
         PRINT "Grade #"; Y " for student " studentName " is " gradeAmount  
         Y = Y + 1  
    WEND  
NEXT X

In this program, the FOR loop specifies that the data will be retrieved for three students.

Graphic

*The relevant pieces of code are  
  
FOR X = 1 to 3  
and  
NEXT X*

Code

FOR X = 1 to 3  
    'write code to get the studentName from the array  
    PRINT "Student "; studentName " grades"  
    Y = 1  
    WHILE Y <= 5  
         'write code to get the gradeAmount from the array  
         PRINT "Grade #"; Y " for student " studentName " is " gradeAmount  
         Y = Y + 1  
    WEND  
NEXT X

When the program begins its execution, the first student's name is retrieved from the array. The name will also appear on screen.

Graphic

*The relevant code is:  
  
'write code to get the studentName from the array  
PRINT "Student " studentName " grades"*

Code

FOR X = 1 to 3  
    'write code to get the studentName from the array  
    PRINT "Student "; studentName " grades"  
    Y = 1  
    WHILE Y <= 5  
         'write code to get the gradeAmount from the array  
         PRINT "Grade #"; Y " for student " studentName " is " gradeAmount  
         Y = Y + 1  
    WEND  
NEXT X

Because the WHILE loop doesn't count in the same way as a FOR loop, the variable Y has to be set to 1 before the WHILE loop starts its execution for each increment of the FOR loop. The value of Y is also manually incremented inside of the WHILE loop.

Graphic

*The relevant pieces of code are:  
  
Y = 1  
and  
Y = Y + 1*

Code

FOR X = 1 to 3  
    'write code to get the studentName from the array  
    PRINT "Student "; studentName " grades"  
    Y = 1  
    WHILE Y <= 5  
         'write code to get the gradeAmount from the array  
         PRINT "Grade #"; Y " for student " studentName " is " gradeAmount  
         Y = Y + 1  
    WEND  
NEXT X

The WHILE loop will read and store values for the student grades until the value of Y equals 6. Even though only five grades are being entered, the value of Y has to be 6 before the loop can break.

Graphic

*The relevant code is:  
  
WHILE Y <= 5  
        'write code to get the gradeAmount from the array  
        PRINT "Grade #"; Y " for student " studentName " is " gradeAmount  
        Y = Y + 1*

Code

FOR X = 1 to 3  
    'write code to get the studentName from the array  
    PRINT "Student "; studentName " grades"  
    Y = 1  
    WHILE Y <= 5  
         'write code to get the gradeAmount from the array  
         PRINT "Grade #"; Y " for student " studentName " is " gradeAmount  
         Y = Y + 1  
    WEND  
NEXT X

When this happens the WHILE loop will break and return control to the loop, executing the FORloop again until the value of X equals 3 or end the program as soon as X is equal to 3.

Graphic

*The relevant code is:   
  
FOR X = 1 to 3  
Here is the code again*

Code

FOR X = 1 to 3  
    'write code to get the studentName from the array  
    PRINT "Student "; studentName " grades"  
    Y = 1  
    WHILE Y <= 5  
         'write code to get the gradeAmount from the array  
         PRINT "Grade #"; Y " for student " studentName " is " gradeAmount  
         Y = Y + 1  
    WEND  
NEXT X

When the program runs, this is an example of the expected output. There are three students listed, with five grades listed for each student.

Code

FOR X = 1 to 3  
    'write code to get the studentName from the array  
    PRINT "Student "; studentName " grades"  
    Y = 1  
    WHILE Y <= 5  
         'write code to get the gradeAmount from the array  
         PRINT "Grade #"; Y " for student " studentName " is " gradeAmount  
         Y = Y + 1  
    WEND  
NEXT X

Question

When will this WHILE loop stop executing?

**Code**  
answer$='Y'  
WHILE answer$ ='Y'  
    PRINT name$  
    PRINT address$  
    PRINT "Would you like to print another ticket? Y/N"  
    INPUT answer$  
WEND

**Options:**

1. When a user enters the value N
2. When a user enters the value Y
3. After it has completed a specified number of iterations
4. It will never stop
5. It will never execute
6. Having any input except Y for answer

Answer

***Option 1:****Correct. Once the input stored in the answer variable is N, the loop condition will be false and the loop will terminate. The next instruction after the loop will be processed.*

***Option 2:****Incorrect. If the input for answer is Y, the WHILE loop will continue executing because the loop condition will still be true.*

***Option 3:****Incorrect. There is no value specified for the number of iterations of the loop. It will continue executing until its tested condition evaluates as false.*

***Option 4:****Incorrect. The loop isn't infinite because it has a condition that can be evaluated to false.*

***Option 5:****Incorrect. The loop has an initial tested condition that evaluates to true. This ensures that the loop will execute at least once.*

***Option 6:****Correct. The loop will terminate if the value input is any value other than Y. N is used as the preferred value to indicate the response is No.*

**Correct answer(s):**

1. When a user enters the value N  
6. Having any input except Y for answer

**2. Summary**

In a program, you may need to test a condition a number of times that will be specified only when the program runs. You can use a WHILE loop to do this. The WHILE loop will iterate as long as its test condition evaluates to true. Once the condition is false, the loop terminates. WHILE loop syntax is similar in Pascal, BASIC, and C.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a03_it_enus/output/html/course_transcript.html#top)

Solving a Repetition Problem

Learning Objectives

*After completing this topic, you should be able to*

* *recognize where to use the WHILE loop*
* *analyze the use of a WHILE loop in PASCAL code in given examples*

**1. When to use WHILE loops**

You can use a WHILE loop to repeat a process as many times as necessary, without having to write long, time-consuming code.  
  
For example, you might be collecting cash donations for a charity and want to print a receipt every time a donation is received. You may also need to calculate the number of donations, the average donation, and the total donations. You can use a WHILE loop to repeat required processes for each donation you receive.

There are four steps you need to follow to create your program:

* analyze the problem statement
* create a defining table
* create an algorithm in pseudocode, and
* convert the pseudocode into an appropriate programming language

Once you've analyzed the problem statement, you can break it down into a defining table that identifies the relevant inputs, processes, and outputs.

Graphic

*The table has three columns: INPUTS, PROCESSES, and OUTPUTS. In the INPUTS column there is one row with the value donation\_amount.  
  
In the PROCESSES column, there are four rows: GET donation\_amount, CALCULATE no\_of\_donations, CALCULATE total\_donations, and CALCULATE average\_donations.  
  
In the OUTPUTS column, there are four rows: receipt, no\_of\_donations, total\_donations, and average\_donations.*

Inputs and outputs in the problem statement are usually indicated by nouns, while process tasks are usually indicated by verbs.

Graphic

*There is a table with three columns labeled Input, Process, and Output.*

**Input**

The amount of the donation received goes in the Input column. This is the basic data that will be processed to get results, or output.  
*The Input column contains donation\_amount*

**Process**

The Process column contains the calculations that need to be done in order to get an output result. These include the calculations for the total donations and the number of donations, as well as the statement to get the source donation amounts.  
*The Process column contains GET donation\_amount, CALCULATE average\_donation, CALCULATE total\_donations, and CALCULATE no\_of\_donations.*

**Output**

The Output column contains the outputs from the calculations in the Process column, which include the results for the total number of donations and the average donation. It also contains the statement to issue a receipt after the amount of the donation is processed.  
*The Output column contains statements for receipt, no\_of\_donations, total\_donation, and average donation.*

You can use the defining table as the basis for writing a pseudocode algorithm, which can then be coded.

You're first prompted by the algorithm to enter the amount for the first donation. It then prints a receipt for that amount.

Code

PRINT Enter the donation  
GET donation\_amount  
PRINT receipt

The number of donations counter is initialized to one, and the total donations amount is set to equal the value of the first donation.

Graphic

*The relevant code is  
  
no\_of\_donations=1*

Code

PRINT Enter the donation  
GET donation\_amount  
PRINT receipt  
no\_of\_donations=1  
total\_donations=donation\_amount

The algorithm then asks if there are further donations to be entered. If the answer is YES, the first iteration of the WHILE loop is processed.

Graphic

*The relevant code is   
  
WHILE other\_donations=YES*

Code

PRINT Enter the donation  
GET donation\_amount  
PRINT receipt  
no\_of\_donations=1  
total\_donations=donation\_amount   
GET other\_donations  
WHILE other\_donations=YES

The compound statement for the WHILE loop performs four steps:

Code

PRINT Enter the donation  
GET donation\_amount  
PRINT receipt  
no\_of\_donations=1  
total\_donations=donation\_amount  
GET other\_donations  
WHILE other\_donations=YES  
    GET donation\_amount  
    PRINT receipt  
    total\_donations=total\_donations+donation\_amount  
    no\_of\_donations=no\_of\_donations+1  
    PRINT 'Are there any more donations to accept?'  
    GET other donations  
ENDWHILE

* it gets the amount for each donation   
  *The relevant code is  
    
  GET donation\_amount.*   
    
  **Code**  
  PRINT Enter the donation  
  GET donation\_amount  
  PRINT receipt  
  no\_of\_donations=1  
  total\_donations=donation\_amount  
  GET other\_donations  
  WHILE other\_donations=YES  
      GET donation\_amount  
      PRINT receipt  
      total\_donations=total\_donations+donation\_amount  
      no\_of\_donations=no\_of\_donations+1  
      PRINT 'Are there any more donations to accept?'  
      GET other donations  
  ENDWHILE
* it prints a receipt   
  *The relevant code is   
    
  PRINT receipt*   
    
  **Code**  
  PRINT Enter the donation  
  GET donation\_amount  
  PRINT receipt  
  no\_of\_donations=1  
  total\_donations=donation\_amount  
  GET other\_donations  
  WHILE other\_donations=YES  
      GET donation\_amount  
      PRINT receipt  
      total\_donations=total\_donations+donation\_amount  
      no\_of\_donations=no\_of\_donations+1  
      PRINT 'Are there any more donations to accept?'  
      GET other donations  
  ENDWHILE
* it adds each donation received to the donations total, and   
  *The relevant code is  
  total\_donations=total\_donations+donation\_amount*   
    
  **Code**  
  PRINT Enter the donation  
  GET donation\_amount  
  PRINT receipt  
  no\_of\_donations=1  
  total\_donations=donation\_amount  
  GET other\_donations  
  WHILE other\_donations=YES  
      GET donation\_amount  
      PRINT receipt  
      total\_donations=total\_donations+donation\_amount  
      no\_of\_donations=no\_of\_donations+1  
      PRINT 'Are there any more donations to accept?'  
      GET other donations  
  ENDWHILE
* it increases the total number of donations by one   
  *The relevant code is  
    
  no\_of\_donations=no\_of\_donations+1*   
    
  **Code**  
  PRINT Enter the donation  
  GET donation\_amount  
  PRINT receipt  
  no\_of\_donations=1  
  total\_donations=donation\_amount  
  GET other\_donations  
  WHILE other\_donations=YES  
      GET donation\_amount  
      PRINT receipt  
      total\_donations=total\_donations+donation\_amount  
      no\_of\_donations=no\_of\_donations+1  
      PRINT 'Are there any more donations to accept?'  
      GET other donations  
  ENDWHILE

The WHILE loop then asks if there are any more donations to accept. If the answer is YES, it continues to execute. The WHILE loop terminates if the user types anything other than YES.

Graphic

*The relevant code is   
  
PRINT 'Are there any more donations to accept?'*

Code

PRINT Enter the donation  
GET donation\_amount  
PRINT receipt  
no\_of\_donations=1  
total\_donations=donation\_amount  
GET other\_donations  
WHILE other\_donations=YES  
    GET donation\_amount  
    PRINT receipt  
    total\_donations=total\_donations+donation\_amount  
    no\_of\_donations=no\_of\_donations+1  
    PRINT 'Are there any more donations to accept?'  
    GET other donations  
ENDWHILE

When the WHILE loop terminates, the total donations are divided by the number of donations to calculate the average donation.

Graphic

*The relevant code is  
average\_donation=total\_donation/no\_of\_donations*

Code

PRINT Enter the donation  
GET donation\_amount  
PRINT receipt  
no\_of\_donations=1  
total\_donations=donation\_amount  
GET other\_donations  
WHILE other\_donations=YES  
    GET donation\_amount  
    PRINT receipt  
    total\_donations=total\_donations+donation\_amount  
    no\_of\_donations=no\_of\_donations+1  
    PRINT 'Are there any more donations to accept?'      
    GET other donations  
ENDWHILE  
average\_donation=total\_donation/no\_of\_donations  
PRINT no\_of\_donations  
PRINT 'the total amount donated is'  
PRINT total\_donation  
PRINT 'the average amount donated is'  
PRINT average donation

The algorithm then displays the number of donations, the average donation, and the total amount donated.

Graphic

*The relevant code is  
  
PRINT no\_of\_donations  
PRINT 'the total amount donated is'  
PRINT total\_donation  
PRINT 'the average amount donated is'  
PRINT average donation*

Code

PRINT Enter the donation  
GET donation\_amount  
PRINT receipt  
no\_of\_donations=1  
total\_donations=donation\_amount  
GET other\_donations  
WHILE other\_donations=YES  
    GET donation\_amount  
    PRINT receipt  
    total\_donations=total\_donations+donation\_amount  
    no\_of\_donations=no\_of\_donations+1  
    PRINT 'Are there any more donations to accept?'      
    GET other donations  
ENDWHILE  
average\_donation=total\_donation/no\_of\_donations  
PRINT no\_of\_donations  
PRINT 'the total amount donated is'  
PRINT total\_donation  
PRINT 'the average amount donated is'  
PRINT average donation

Question

You've created an algorithm to count the number of logs placed on a logging truck. Once the truck is full of logs, it can leave and an empty truck can take its place.  
  
Which type of loop is it most appropriate to use in your program?

**Options:**

1. WHILE
2. FOR
3. IF
4. CASE

Answer

***Option 1:****Correct. A WHILE loop allows processing to continue as long as a tested condition is true, once the condition is false, the loop will end. In this instance as long as the truck can be loaded, the loop will execute.*

***Option 2:****Incorrect. When you use a FOR loop, you generally need to know how many times you want the loop to execute. In this instance, the number of logs that can go on a truck is unknown, so a FOR loop is unsuitable.*

***Option 3:****Incorrect. IF is a statement, not a loop. It's most often used as a selection statement in programming.*

***Option 4:****Incorrect. CASE is a statement, not a loop. It commonly replaces nested IFstatements.*

**Correct answer(s):**

1. WHILE

Question

Sequence the lines in the pseudocode algorithm.

**Options:**

1. GET numLogsOnTruck
2. WHILE numLogsOnTruck < 50
3. PUT another log on the truck  
   numLogsOnTruck = numLogsOnTruck + 1
4. ENDWHILE
5. DO THIS  
   Send truck on its way  
   Receive an empty truck

Answer

**Correct answer(s):**

**GET numLogsOnTruck is ranked**

You first have to determine the input you need to base your calculations on. This is the first line of the algorithm

**WHILE numLogsOnTruck < 50 is ranked**

You set the initial condition to be executed to begin running the loop. This is the second line of the algorithm

**PUT another log on the truck  
numLogsOnTruck = numLogsOnTruck + 1 is ranked**

The counter is set to increment the number of logs on the truck once for each iteration. These are the third and fourth lines of the algorithm.

**ENDWHILE is ranked**

You close the WHILE loop. This is the fifth line of the algorithm.

**DO THIS  
Send truck on its way  
Receive an empty truck is ranked**

Once the conditions in the loop are met to indicate the truck is full, it can leave and be replaced by an empty truck. This is the sixth and final line of the algorithm.

**2. Using a WHILE loop in Pascal**

Once you've written the algorithm for collecting donations in pseudocode, you can code the algorithm – for example, in Pascal. The program starts by performing six actions:

Code

writeln ('Enter the first donation');  
read (donation\_amount);  
print ('Thank you for your donation of', donation\_amount);   
no\_of\_donations := 1;  
total\_donations := donation\_amount;  
writeln ('Are there any more donations to accept? Y/N');  
read (other\_donations);

* prompting the user to enter the first donation   
  *The relevant code is   
    
  writeln ('Enter the first donation');*   
    
  **Code**  
  writeln ('Enter the first donation');  
  read (donation\_amount);  
  print ('Thank you for your donation of', donation\_amount);   
  no\_of\_donations := 1;  
  total\_donations := donation\_amount;  
  writeln ('Are there any more donations to accept? Y/N');  
  read (other\_donations);
* reading the donation amount   
  *The relevant code is   
    
  read (donation\_amount);*   
    
  **Code**  
  writeln ('Enter the first donation');  
  read (donation\_amount);  
  print ('Thank you for your donation of', donation\_amount);   
  no\_of\_donations := 1;  
  total\_donations := donation\_amount;  
  writeln ('Are there any more donations to accept? Y/N');  
  read (other\_donations);
* printing a receipt   
  *The relevant code is   
    
  print ('Thank you for your donation of', donation\_amount);*   
    
  **Code**  
  writeln ('Enter the first donation');  
  read (donation\_amount);  
  print ('Thank you for your donation of', donation\_amount);   
  no\_of\_donations := 1;  
  total\_donations := donation\_amount;  
  writeln ('Are there any more donations to accept? Y/N');  
  read (other\_donations);
* initializing the number of donations to one   
  *The relevant code is  
    
  no\_of\_donations := 1;*   
    
  **Code**  
  writeln ('Enter the first donation');  
  read (donation\_amount);  
  print ('Thank you for your donation of', donation\_amount);   
  no\_of\_donations := 1;  
  total\_donations := donation\_amount;  
  writeln ('Are there any more donations to accept? Y/N');  
  read (other\_donations);
* assigning the initial donation amount to total donations, and   
  *The relevant code is  
    
  total\_donations := donation\_amount;*   
    
  **Code**  
  writeln ('Enter the first donation');  
  read (donation\_amount);  
  print ('Thank you for your donation of', donation\_amount);   
  no\_of\_donations := 1;  
  total\_donations := donation\_amount;  
  writeln ('Are there any more donations to accept? Y/N');  
  read (other\_donations);
* asking for and accepting input for more donations   
  *The relevant code is   
    
  writeln ('Are there any more donations to accept? Y/N');  
  read (other\_donations);*   
    
  **Code**  
  writeln ('Enter the first donation');  
  read (donation\_amount);  
  print ('Thank you for your donation of', donation\_amount);   
  no\_of\_donations := 1;  
  total\_donations := donation\_amount;  
  writeln ('Are there any more donations to accept? Y/N');  
  read (other\_donations);

The entry condition of the WHILE loop is decided by the value of the other\_donationsvariable.

Graphic

*The relevant code is   
  
read (other\_donations);*

Code

writeln ('Enter the first donation');  
read (donation\_amount);  
print ('Thank you for your donation of', donation\_amount);   
no\_of\_donations := 1;  
total\_donations := donation\_amount;  
writeln ('Are there any more donations to accept? Y/N');  
read (other\_donations);

Adding do to the while loop requires the loop to run once and check if the conditions are met to return a value of 'Y'.

Code

writeln ('Enter the first donation');  
read (donation\_amount);  
print ('Thank you for your donation of', donation\_amount);   
no\_of\_donations := 1;  
total\_donations := donation\_amount;  
writeln ('Are there any more donations to accept? Y/N');  
read (other\_donations);  
while other\_donations = 'Y' do

If a 'Y' value is returned, the while loop will run and will continue to execute for as many times as the 'Y' value is returned.

Code

writeln ('Enter the first donation');  
read (donation\_amount);  
print ('Thank you for your donation of', donation\_amount);   
no\_of\_donations := 1;  
total\_donations := donation\_amount;  
writeln ('Are there any more donations to accept? Y/N');  
read (other\_donations);  
while other\_donations = 'Y' do

begin and end tags are used to contain the statements within the loop.

Code

writeln ('Enter the first donation');  
read (donation\_amount);  
print ('Thank you for your donation of', donation\_amount);   
no\_of\_donations := 1;  
total\_donations := donation\_amount;  
writeln ('Are there any more donations to accept? Y/N');   
read (other\_donations);  
while other\_donations = 'Y' do  
begin  
    read (donation\_amount);  
    print ('Thank you for your donation of', donation\_amount);  
    no\_of\_donations := no\_of\_donations + 1;  
    total\_donation := total\_donation + donation\_amount;  
    writeln ('Are there any more donations to accept? Y/N') ;  
    read (other\_donations)  
end;  
average\_donation := total\_donation/no\_of\_donations;

So that the loop doesn't run indefinitely, the entry condition other\_donations has a new value read at the end of the loop. The loop terminates when the other\_donations variable no longer contains the value 'Y'.

Graphic

*The relevant code is   
  
Y/N*

Code

writeln ('Enter the first donation');  
read (donation\_amount);  
print ('Thank you for your donation of', donation\_amount);   
no\_of\_donations := 1;  
total\_donations := donation\_amount;  
writeln ('Are there any more donations to accept? Y/N');   
read (other\_donations);  
while other\_donations = 'Y' do  
begin  
    read (donation\_amount);  
    print ('Thank you for your donation of', donation\_amount);  
    no\_of\_donations := no\_of\_donations + 1;  
    total\_donation := total\_donation + donation\_amount;  
    writeln ('Are there any more donations to accept? Y/N') ;  
    read (other\_donations)  
end;  
average\_donation := total\_donation/no\_of\_donations;

The average donation is calculated once the loop has terminated.

Graphic

*The relevant code is  
  
average\_donation := total\_donation/no\_of\_donations;*

Code

writeln ('Enter the first donation');  
read (donation\_amount);  
print ('Thank you for your donation of', donation\_amount);   
no\_of\_donations := 1;  
total\_donations := donation\_amount;  
writeln ('Are there any more donations to accept? Y/N');   
read (other\_donations);  
while other\_donations = 'Y' do  
begin  
    read (donation\_amount);  
    print ('Thank you for your donation of', donation\_amount);  
    no\_of\_donations := no\_of\_donations + 1;  
    total\_donation := total\_donation + donation\_amount;  
    writeln ('Are there any more donations to accept? Y/N') ;  
    read (other\_donations)  
end;  
average\_donation := total\_donation/no\_of\_donations;

Finally the program displays output, including the number of donations, total amount of the donations, and average donation amount.

Graphic

*The relevant code is  
  
writeln ('The number of donations is ', no\_of\_donations);  
writeln ('The total amount donate is ', total\_donation);  
writeln ('The average amount donated is ', average\_donation);*

Code

writeln ('Enter the first donation');  
read (donation\_amount);  
print ('Thank you for your donation of', donation\_amount);   
no\_of\_donations := 1;  
total\_donations := donation\_amount;  
writeln ('Are there any more donations to accept? Y/N');   
read (other\_donations);  
while other\_donations = 'Y' do  
begin  
    read (donation\_amount);  
    print ('Thank you for your donation of', donation\_amount);  
    no\_of\_donations := no\_of\_donations + 1;  
    total\_donation := total\_donation + donation\_amount;  
    writeln ('Are there any more donations to accept? Y/N') ;  
    read (other\_donations)  
end;  
average\_donation := total\_donation/no\_of\_donations;  
writeln ('The number of donations is ', no\_of\_donations);  
writeln ('The total amount donate is ', total\_donation);  
writeln ('The average amount donated is ', average\_donation);

Question

What is the error in this example of Pascal code?

**Code**  
program loop;  
var i, limit : integer;  
  
begin  
    writeln ('Please enter a limit.');  
    readln (limit);  
    while i <= limit  
    begin do  
        write(i, ' ');  
        i := i + 1  
    end;  
end

**Options:**

1. The keyword do is in the wrong place
2. The begin and end tags around the while loop's statements are optional
3. The writeln statement should come after the while loop

Answer

***Option 1:****Correct. The keyword do should be on the while line, not the begin line.*

***Option 2:****Incorrect. Statements in the while loop require a begin and end tag.*

***Option 3:****Incorrect. The while loop comes after the writeln statement as a value needs to be entered before the loop can execute.*

**Correct answer(s):**

1. The keyword do is in the wrong place

**3. Summary**

A while loop is useful for repeating processes a required number of times. The loop can be designed to execute as long as there is information to process. It will stop executing once an entry condition statement returns a false value when queried at the beginning of an iteration.  
  
In Pascal, you use begin and end tags to enclose the statements in a while loop.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a03_it_enus/output/html/course_transcript.html#top)

Using Loops

Learning Objectives

*After completing this topic, you should be able to*

* *create a defining table to determine inputs, processes, and outputs*
* *structure an algorithm containing a FOR loop*
* *complete a WHILE loop in an algorithm*

**1. Exercise overview**

In this exercise, you're required to use loops in a program.

This involves the following tasks:

* creating a defining table to determine inputs, processes, and outputs
* structuring an algorithm containing a FOR loop, and
* completing a WHILE loop in an algorithm

**2. Creating defining tables**

You are starting to create a program that accepts donation amounts as input and then generates output such as total donations and average amount donated. You can use a defining table to assist you in your planning.

Question

Match each defining table heading with the element contained in its column.

**Options:**

1. Input
2. Process
3. Output

**Targets:**

1. Amount of each donation
2. Calculate the total donations
3. Total donations

Answer

*The value amount of each donation should go into the Input column.*

*The calculation to determine the total donations should go into the Process column.*

*The final value for the total donations should go into the Output column.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Question

Which process task would be carried out first?

**Options:**

1. Get the donation amount
2. Calculate the average donation
3. Calculate the total amount donated
4. Calculate the number of donations

Answer

***Option 1:****Correct. The statement processed first should be the GET pseudocode statement.*

***Option 2:****Incorrect. You can only calculate the average donation once you have totaled the donation amounts and determined the number of donations.*

***Option 3:****Incorrect. The statement processed first should be the GET statement because it retrieves the data for calculations. You can only calculate the total once you have had all the donation amounts input.*

***Option 4:****Incorrect. You can only calculate the number of donations after the donation amounts data has been processed.*

**Correct answer(s):**

1. Get the donation amount

**3. Structuring a FOR loop**

A FOR loop removes the need to repeat code because it allows you to run a task a predetermined number of times.

Code

GET value  
   FOR i = 1 TO value  
       Process  
   ENDFOR

Question

Identify correct statements about the operations in a FOR loop.

**Options:**

1. The counter is initialized before code in the FOR loop is executed
2. After an iteration, the value of the counter is tested against the loop limit
3. When it's initialized, the amount being incremented or decremented always has a starting value of zero
4. The loop terminates before the counter exceeds a set limit

Answer

***Option 1:****Correct. The counter in the loop is initialized before code in the FOR loop is executed.*

***Option 2:****Correct. After each iteration, the counter is compared to the set limit value. The loop will continue to repeat as long as the counter remains within the range set by the loop limit.*

***Option 3:****Incorrect. The amount being incremented or decremented has a default starting value of one, which is then incremented or decremented as the loop executes.*

***Option 4:****Incorrect. The loop stops executing only when the counter exceeds the range set by the limit.*

**Correct answer(s):**

1. The counter is initialized before code in the FOR loop is executed  
2. After an iteration, the value of the counter is tested against the loop limit

**4. Using WHILE loops**

You are writing an algorithm that converts weights from pounds to kilograms. Because you want the algorithm to run as long as there is a valid input for the weight in pounds, you use a WHILE loop.

Code

value = “Y”  
    WHILE   
        value = “Y”   
        Process  
    ENDWHILE

Question

You have completed most of the code for a pseudocode algorithm. You're not sure how many iterations are required, so you need to include a conditional statement.  
  
Which conditional statement should you use in the WHILE loop?

**Code**  
more\_weights="Y"  
    WHILE INSERT THE MISSING CODE      
        GET pounds     
        PRINT pounds     
        kilograms=pounds\*0.4536     
        PRINT 'The weight in kilograms is'     
        PRINT kilograms     
        GET 'more\_weights? Y/N'  
    ENDWHILE

**Options:**

1. more\_weights = "Y"
2. pounds > 20
3. more\_weights = 0

Answer

***Option 1:****Correct. The more\_weights variable is assigned an initial value of "Y" before the loop. This ensures that the loop executes at least once.*

***Option 2:****Incorrect. This option requires the input weight to be greater than 20 pounds. However, there shouldn't be any restriction on the weights that can be converted.*

***Option 3:****Incorrect. If you use this conditional expression, the loop would never run because the more\_weights variable will never be 0. The loop has an initial value of "Y" and with every loop the user enters either "Y" to continue the loop or "N" to terminate it.*

**Correct answer(s):**

1. more\_weights = "Y"

Question

In your algorithm, the code inside the WHILE loop ends with a prompt asking whether there are more weights.   
  
What happens if the highlighted statement is omitted?

*The highlighted statement is GET 'more\_weights =?Y/N'*

**Code**  
more\_weights="Y"  
WHILE more weights = "Y"  
    GET weights  
    PRINT weight\_in\_pounds      
    Kilograms=pounds\*0.4536  
    PRINT 'The weight in kilograms is'  
    PRINT kilograms  
    INPUT 'more\_weights Y/N?' more\_weights  
ENDWHILE

**Options:**

1. The loop runs indefinitely
2. The loop ends after one iteration
3. The loop can't not function, failing to start

Answer

***Option 1:****Correct. Without the closing statement – which asks whether there are more weights – the condition variable more\_weights will never be updated. Consequently, it will always have a value of "Y" and the loop will continue iterating indefinitely.*

***Option 2:****Incorrect. The while condition is evaluated before each iteration. It completes one iteration because the more\_weights variable is assigned an initial value of "Y". The loop would only end after one iteration if the more\_weights variable was set to something other than "Y"during the first iteration.*

***Option 3:****Incorrect. The more\_weights variable is assigned an initial value of "Y" outside the loop. Consequently, the while condition is true when it's initially evaluated and the loop will execute.*

**Correct answer(s):**

1. The loop runs indefinitely

A defining table has been created, a FOR loop algorithm has been written, and WHILE loops have been used in a program.

Code

value = “Y”  
    WHILE   
        value = “Y”   
        Process  
    ENDWHILE

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a03_it_enus/output/html/course_transcript.html#top)

© 2016 Skillsoft Ireland Limited