CONTROL STRUCTURES pdf

1: Water Control Structures | Advanced Drainage Systems

A control structure is a block of programming that analyzes variables and chooses a direction in which to go based on given parameters. The term flow control details the direction the program takes (which way program control "flows"). Hence it is the basic decision-making process in computing; flow.

Jump to navigation Jump to search Flow Control Overview[edit] A control structure is a block of programming that analyzes variables and chooses a direction in which to go based on given parameters. The term flow control details the direction the program takes which way program control "flows". Hence it is the basic decision-making process in computing; flow control determines how a computer will respond when given certain conditions and parameters. Basic Terminologies[edit] Those initial conditions and parameters are called preconditions. Preconditions are the state of variables before entering a control structure. Based on those preconditions, the computer runs an algorithm the control structure to determine what to do. The result is called a postcondition. Postconditions are the state of variables after the algorithm is run. An Example[edit] Let us analyze flow control by using traffic flow as a model. A vehicle is arriving at an intersection. Thus, the precondition is the vehicle is in motion. Suppose the traffic light at the intersection is red. The control structure must determine the proper course of action to assign to the vehicle. The vehicle is in motion. Treatments of Information through Control Structures Is the traffic light green? If so, then the vehicle may stay in motion. Is the traffic light red? If so, then the vehicle must stop. End of Treatment Postcondition: The vehicle comes to a stop. Thus, upon exiting the control structure, the vehicle is stopped. The condition can include a variable, or be a variable. If the variable is an integer 2, it will be true, because any number that is not zero will be true. If the condition is true, then an action occurs. If the condition is false, nothing is done. If the variable indeed holds a value consistent with being true, then the course of action is taken. If the variable is not true, then there is no course of action taken. If the condition is false, take an alternate action. If the variable is false, the control structure calls a routine and completely skips the THEN clause. This structure is useful when performing iterative instructions to satisfy a certain parameter. Note that when the structure quits, it will not execute the Add function: This instruction is useful if a parameter needs to be tested repeatedly before acceptance. A WHILE loop initially checks to see if the parameters have been satisfied before executing an instruction. A for loop usually has three commands. The second is the end condition same as in a while loop and is run every round. The third is also run every round and is usually used to modify a value used in the condition block. FOR X is 0. X is less than Here is a while loop that does the same: X is 0 WHILE X is less than 10 add 1 to X Some programming languages also have a foreach loop which will be useful when working with arrays and collections.

2: Computer programming language - Control structures | www.enganchecubano.com

This chapter shows you how to structure the flow of control through a PL/SQL program. You learn how statements are connected by simple but powerful control structures that have a single entry and exit point. Collectively, these structures can handle any situation. Their proper use leads naturally to.

Graphs and Graph Algorithms 1. Both of these are supported by Python in various forms. The programmer can choose the statement that is most useful for the given circumstance. For iteration, Python provides a standard while statement and a very powerful for statement. The while statement repeats a body of code as long as a condition is true. The condition on the while statement is evaluated at the start of each repetition. If the condition is True, the body of the statement will execute. It is easy to see the structure of a Python while statement due to the mandatory indentation pattern that the language enforces. The while statement is a very general purpose iterative structure that we will use in a number of different algorithms. In many cases, a compound condition will control the iteration. The value of the variable counter would need to be less than or equal to 10 and the value of the variable done would need to be False not False is True so that True and True results in True. Even though this type of construct is very useful in a wide variety of situations, another iterative structure, the for statement, can be used in conjunction with many of the Python collections. The for statement can be used to iterate over the members of a collection, so long as the collection is a sequence. The body of the iteration is then executed. This works for any collection that is a sequence lists, tuples, and strings. A common use of the for statement is to implement definite iteration over a range of values. The range function will return a range object representing the sequence 0,1,2,3,4 and each value will be assigned to the variable item. This value is then squared and printed. The other very useful version of this iteration structure is used to process each character of a string. The following code fragment iterates over a list of strings and for each string processes each character by appending it to a list. The result is a list of all the letters in all of the words. Most programming languages provide two versions of this useful construct: A simple example of a binary selection uses the ifelse statement. If it is, a message is printed stating that it is negative. If it is not, the statement performs the else clause and computes the square root. Selection constructs, as with any control construct, can be nested so that the result of one question helps decide whether to ask the next. For example, assume that score is a variable holding a reference to a score for a computer science test. If the score is greater than or equal to 90, the statement will print A. If it is not else , the next question is asked. If the score is greater than or equal to 80 then it must be between 80 and 89 since the answer to the first question was false. In this case print B is printed. You can see that the Python indentation pattern helps to make sense of the association between if and else without requiring any additional syntactic elements. An alternative syntax for this type of nested selection uses the elif keyword. The else and the next if are combined so as to eliminate the need for additional nesting levels. Note that the final else is still necessary to provide the default case if all other conditions fail. With this statement, if the condition is true, an action is performed. In the case where the condition is false, processing simply continues on to the next statement after the if. For example, the following fragment will first check to see if the value of a variable n is negative. If it is, then it is modified by the absolute value function. Regardless, the next action is to compute the square root. Modify the code from Activecode 8 so that the final list only contains a single copy of each letter. A list comprehension allows you to easily create a list based on some processing or selection criteria. For example, if we would like to create a list of the first 10 perfect squares, we could use a for statement: The general syntax for a list comprehension also allows a selection criteria to be added so that only certain items get added. Any sequence that supports iteration can be used within a list comprehension to construct a new list. For an extra challenge, see if you can figure out how to remove the duplicates. Created using Runestone 2.

3: Nested Control Structures (Visual Basic) | Microsoft Docs

A control structure is a primary concept in most high-level programming languages. In its simplest sense, it is a block of code. More specifically, control structures are blocks of code that dictate the flow of control.

Several programming languages e. Scala has for-expressions , which generalise collection-controlled loops, and also support other uses, such as asynchronous programming. Haskell has do-expressions and comprehensions, which together provide similar function to for-expressions in Scala. Where a more specific looping construct can be used, it is usually preferred over the general iteration construct, since it often makes the purpose of the expression clearer. Infinite loop Infinite loops are used to assure a program segment loops forever or until an exceptional condition arises, such as an error. For instance, an event-driven program such as a server should loop forever, handling events as they occur, only stopping when the process is terminated by an operator. Infinite loops can be implemented using other control flow constructs. Most commonly, in unstructured programming this is jump back up goto , while in structured programming this is an indefinite loop while loop set to never end, either by omitting the condition or explicitly setting it to true, as while true Some languages have special constructs for infinite loops, typically by omitting the condition from an indefinite loop. Examples include Ada loop Often, an infinite loop is unintentionally created by a programming error in a condition-controlled loop, wherein the loop condition uses variables that never change within the loop. Continuation with next iteration[edit] Sometimes within the body of a loop there is a desire to skip the remainder of the loop body and continue with the next iteration of the loop. Some languages provide a statement such as continue most languages , skip, or next Perl and Ruby , which will do this. The effect is to prematurely terminate the innermost loop body and then resume as normal with the next iteration. If the iteration is the last one in the loop, the effect is to terminate the entire loop early. Redo current iteration[edit] Some languages, like Perl and Ruby, have a redo statement that restarts the current iteration from the start. Ruby has a retry statement that restarts the entire loop from the initial iteration. Early exit from loops[edit] When using a count-controlled loop to search through a table, it might be desirable to stop searching as soon as the required item is found. Some programming languages provide a statement such as break most languages , Exit Visual Basic , or last Perl , which effect is to terminate the current loop immediately, and transfer control to the statement immediately after that loop. The following example is done in Ada which supports both early exit from loops and loops with test in the middle. Both features are very similar and comparing both code snippets will show the difference: Text IO; with Ada. Some languages support breaking out of nested loops; in theory circles, these are called multi-level breaks. One common use example is searching a multi-dimensional table. This can be done either via multilevel breaks break out of N levels , as in bash [6] and PHP, [7] or via labeled breaks break out and continue at given label , as in Java and Perl. C does not include a multilevel break, and the usual alternative is to use a goto to implement a labeled break. Rao Kosaraju refined the structured program theorem by proving that it is possible to avoid adding additional variables in structured programming, as long as arbitrary-depth, multi-level breaks from loops are allowed. There are other proposed control structures for multiple breaks, but these are generally implemented as exceptions instead. Watt notes that a class of sequencers known as escape sequencers, defined as "sequencer that terminates execution of a textually enclosing command or procedure", encompasses both breaks from loops including multi-level breaks and return statements. As commonly implemented, however, return sequencers may also carry a return value, whereas the break sequencer as implemented in contemporary languages usually cannot. Loop variants are used to guarantee that loops will terminate. A loop invariant is an assertion which must be true before the first loop iteration and remain true after each iteration. This implies that when a loop terminates correctly, both the exit condition and the loop invariant are satisfied. Loop invariants are used to monitor specific properties of a loop during successive iterations. Some programming languages, such as Eiffel contain native support for loop variants and invariants. Loop sublanguage[edit] Some Lisp dialects provide an extensive sublanguage for describing Loops. An early example can be found in Conversional Lisp of Interlisp. Common Lisp [15] provides a Loop macro which implements such a sublanguage. Loop system cross-reference table[edit] It has been suggested that this article be split into a new article titled Comparison of programming languages control flow.

4: Timewell Drainage Products | Water Control Structures

The three basic control structures in virtually every procedural language are: Sequence is the default control structure; instructions are executed one after another. They might, for example, carry out a series of arithmetic operations, assigning results to variables, to find the roots of a quadratic equation ax2 + bx + c = 0.

If it had been anything other than 2 it would now be 0. For example, some browsers provide document. In order to solve this problem, we could write this: This is known as a short-circuit. The operator has a similar feature, but it will only evaluate the second test if the first one fails. This may seem convenient, as it allows you to make your code a tiny bit shorter, but I recommend avoiding this syntax. It makes your code harder to read, especially if you start nesting your control structures. It also makes it easy to forget to put them in when you needed them, and also makes debugging code much harder, since you will need to go back through your code to add them so that you can add extra debugging tests. It is best to always use the curly braces, even if they are optional. As always, there is an exception. Typically, it is used to cycle through the contents of an array, or to create a specific number of new objects, but it can do many more useful things if needed. As with all variables, you must declare them if you have not done so already. You can define multiple variables if needed, using: Typically, this is used to increment or decrement a stepping variable, and it is possible to perform actions on more than one variable by separating them with a comma: Every time you create properties or methods on an object, these will be added to the list of properties that will be exposed. Most internal properties the ones that JavaScript creates will also be exposed, but JavaScript engines are allowed to hide internal properties and methods if they want to. You should not rely on any specific behaviour here, but note that some browsers will give the internal properties and methods of intrinsic objects, and some will not. Again, you should declare the variable names that you use, if you have not done so already. Each time it loops, it assigns the next property name as a string value to myVariable. You can then use array notation to access the value of that property. The following example writes all the exposed properties of the document object: It is very easy to make mistakes here, so be careful not to mistake these property types for each other. It will continue to run as long as the condition is satisfied: Firstly, it would use the value of myVariable to index the array cell, then it would increment myVariable. Firstly, it would increment the value of myVariable, then it would use the new value to index the array cell. If I had done this, myArray[2] to myArray[6] would now be 1. These features also work outside loops, but this is where you will most commonly see them, so I have included them here. The condition is evaluated at the end of the loop, meaning that even if the condition is never satisfied, it will still run through the loop at least once. The solution is to use break; as follows The use of the break statement is described below. This makes it superior to the original way of handling script errors without error messages where scripts are completely aborted: Thankfully these browsers are hardly used any more. It would also be useful for checking for stupid bugs, like where checking for something like navigator. However, the error is not correctly thrown for these errors. Unfortunately, if you use this structure in any script run by a browser that does not support it, the browser will abort the entire script with errors, even if it does not use the part containing the structure. Thankfully, these old browsers can be safely ignored. It should never be used to detect if a browser supports a method or property like document. So when should it be used? It can be used for W3C DOM scripting, where you may want to avoid DOM mutation errors for example , which are valid errors, but serve to warn you not to do something, and do not always need to abort the whole script. However, they will still run the script it is not possible to protect them by using the language attribute on the script tag, as you need to use JavaScript 1. This means that the older browsers will still produce errors, unless you define the old error handling method in an earlier script. It can be used to check if accessing a frameset frame will cause a browser security error for example, if the page in the frame belongs to another site. It could also enable you to avoid problems where different browsers support the same methods but expect a different syntax, for example, the selectBox. This is a very important rule to learn, as it forms the basis of object and capability detection, and is fundamental to making cross browser scripts. This will be true if: Netscape 4, Internet Explorer 5. It can also be used to check for the existence of named properties of an object. In most cases, it is best to use a conditional without a condition, as shown above. An example would be where you want to check for the existence of a property whose value may be 0, or an empty string, or null. If you know what the type of the property will be, it is possible to achieve this using identity operators, or the typeof operator, as shown here: This allows you to test for its existence, no matter what value it currently holds, and no matter what type of value it currently has even if it has been assigned a value of undefined. This limits the usefulness a little, as it can only search for the name, and cannot be used to see if one of the properties holds a specific value or value type. In most other browsers, the two alternatives perform about the same. Assignments inside a conditional JavaScript allows you to perform an assignment at the same time as testing if the assignment worked. The syntax used is: Control Structure Labels are very rarely used in JavaScript. The break statement Writing break inside a switch, for, for-in, while or do - while control structure will cause the program to jump to the end of the statement. If you just use, for example: But if you use this: If you just use this, for example: If you use this instead:

5: Statements and flow control - C++ Tutorials

As we noted earlier, algorithms require two important control structures: iteration and selection. Both of these are supported by Python in various forms. The programmer can choose the statement that is most useful for the given circumstance. For iteration, Python provides a standard while statement.

Once the expression results false, the program will skip beyong the while loop and move on. Since the do-while loop always checks the condition of the expression last, the do-while will ALWAYS run the statments at least once. It will run for 3 bursts before running the code after the structure, stopping the motors. It will always run for at least 1 three second burst before checking to see the condition of the expression. It is a neat and compact solution to looping. It will repeat this process 4 times, completing a square. If the else clause is present and the expresssion is zero false , control will pass to statement 2. The else part is optional, and if absent, a false expression will simply result in skipping over the statement 1. An else always matches the nearest previous unmatched if; braces may be used to override this when necessary, or for clarity. The Substatement controlled by a switch is typically compound. Any statement within the substatement may be labeled with one or more case labels, which consist of the keyword case followed by a constant expression and then a colon: No two of the case constants associated with the same switch may have the same value. There may be at most one default label associated with a switch - if none of the case labels are equal to the expression in the parentheses following switch, control passes to the default label, or if there is no default label, execution resumes just beyond the entire construct. Switches may be nested; a case or default label is associated with the innermost switch that contains it. Switch statements can "fall through", that is, when one case section has completed its execution, statements will continue to be executed downward until a break; statement is encountered. Fall-through is useful in some circumstances, but is usually not desired. In the preceding example, if label2 is reached, the statements statements 2 are executed and nothing more inside the braces. However if label1 is reached, both statements 1 and statements 2 are executed since there is no break to separate the two case statements. There is no fall through in this switch statement as each case has a break to jump out of the switch once that case is finished. Since the while loop always checks the condition of the expression first, it is possible that the while loop never runs the statements within the braces. Once it reaches 20 it will skip the loop, running the code after the structure, stopping the motors. This program would drive your robot forward until it was 20 units away from an object.

6: The Three Fundamental Control Structures â€“ Aristides S. Bouras

In computer science, control flow (or flow of control) is the order in which individual statements, instructions or function calls of an imperative program are executed or evaluated. The emphasis on explicit control flow distinguishes an imperative programming language from a declarative programming language.

The intent of this guide is to reduce cognitive friction when scanning code from different authors. It does so by enumerating a shared set of rules and expectations about how to format PHP code. The style rules herein are derived from commonalities among the various member projects. When various authors collaborate across multiple projects, it helps to have one set of guidelines to be used among all those projects. Thus, the benefit of this guide is not in the rules themselves, but in the sharing of those rules. Code MUST use 4 spaces for indenting, not tabs. Example This example encompasses some of the rules below as a quick overview: Blank lines MAY be added to improve readability and to indicate related blocks of code. Using only spaces, and not mixing spaces with tabs, helps to avoid problems with diffs, patches, history, and annotations. The use of spaces also makes it easy to insert fine-grained sub-indentation for inter-line alignment. When present, all use declarations MUST go after the namespace declaration. There MUST be one use keyword per declaration. There MUST be one blank line after the use block. A property declaration looks like the following. A method declaration looks like the following. Note the placement of parentheses, commas, spaces, and braces: Method arguments with default values MUST go at the end of the argument list. When the argument list is split across multiple lines, the closing parenthesis and opening brace MUST be placed together on their own line with one space between them. When present, the static declaration MUST come after the visibility declaration. Control Structures The general style rules for control structures are as follows: This standardizes how the structures look, and reduces the likelihood of introducing errors as new lines get added to the body. Note the placement of parentheses, spaces, and braces; and that else and elseif are on the same line as the closing brace from the earlier body. Note the placement of parentheses, spaces, and braces. The case statement MUST be indented once from switch, and the break keyword or other terminating keyword MUST be indented at the same level as the case body. Closures Closures MUST be declared with a space after the function keyword, and a space before and after the use keyword. Closure arguments with default values MUST go at the end of the argument list. A closure declaration looks like the following. When the ending list whether of arguments or variables is split across multiple lines, the closing parenthesis and opening brace MUST be placed together on their own line with one space between them. The following are examples of closures with and without argument lists and variable lists split across multiple lines. Conclusion There are many elements of style and practice intentionally omitted by this guide. These include but are not limited to: Declaration of global variables and global constants Declaration of functions.

7: Control Structures

The iterative control structures are used for repetitively executing a block of code multiple times; the following are the basic iterative control structures. â€¢ for: A for loop execute the statements for a specific number of times mentioned in the condition.

They can be combined in any way necessary to deal with a given problem. Figure Control Structures The selection structure tests a condition, then executes one sequence of statements instead of another, depending on whether the condition is true or false. The iteration structure executes a sequence of statements repeatedly as long as a condition holds true. The sequence structure simply executes a sequence of statements in the order in which they occur. IF Statements Often, it is necessary to take alternative actions depending on circumstances. The IF statement lets you execute a sequence of statements conditionally. That is, whether the sequence is executed or not depends on the value of a condition. There are three forms of IF statements: If the condition is false or null, the IF statement does nothing. In either case, control passes to the next statement. Thus, the ELSE clause ensures that a sequence of statements is executed. That is, IF statements can be nested, as the following example shows: Conditions are evaluated one by one from top to bottom. If any condition is true, its associated sequence of statements is executed and control passes to the next statement. If all conditions are false or null, the sequence in the ELSE clause is executed. Consider the following example: Nevertheless, bonus is assigned the proper value of because the second condition is never tested. Guidelines Avoid clumsy IF statements like those in the following example: First, the value of a Boolean expression can be assigned directly to a Boolean variable. So, you can replace the first IF statement with a simple assignment, as follows: So, you can simplify the condition in the second IF statement, as follows: That way, your code will be easier to read and understand. Compare the following IF statements: There are three forms of LOOP statements: If further processing is undesirable or impossible, you can use an EXIT statement to complete the loop. You can place one or more EXIT statements anywhere inside a loop, but nowhere outside a loop. There are two forms of EXIT statements: When an EXIT statement is encountered, the loop completes immediately and control passes to the next statement. If the condition is true, the loop completes and control passes to the next statement after the loop. So, a statement inside the loop must change the value of the condition. For example, compare the following statements: The label, an undeclared identifier enclosed by double angle brackets, must appear at the beginning of the LOOP statement, as follows: With either form of EXIT statement, you can complete not only the current loop, but any enclosing loop. Simply label the enclosing loop that you want to complete. Then, use the label in an EXIT statement, as follows: If the condition is true, the sequence of statements is executed, then control resumes at the top of the loop. If the condition is false or null, the loop is bypassed and control passes to the next statement. The condition is tested at the top of the loop, so the sequence might execute zero times. In the last example, if the initial value of total is larger than , the condition is false and the loop is bypassed. Therefore, the sequence of statements is executed at least once. Otherwise, you have an infinite loop. For example, the following LOOP statements are logically equivalent: FOR loops iterate over a specified range of integers. As the next example shows, the sequence of statements is executed once for each integer in the range. After each iteration, the loop counter is incremented. FOR i IN However, as the example below shows, if you use the keyword REVERSE, iteration proceeds downward from the higher bound to the lower bound. After each iteration, the loop counter is decremented. Nevertheless, you write the range bounds in ascending not descending order. FOR ctr IN The lower bound need not be 1, as the examples below show. However, the loop counter increment or decrement must be 1. Inside the FOR loop, simply multiply each reference to the loop counter by the new increment. What happens if the lower bound of a loop range evaluates to a larger integer than the upper bound? As the next example shows, the sequence of statements within the loop is not executed and control passes to the next statement: You cannot reference it outside the loop. After the loop is exited, the loop counter is undefined, as the following example shows: The next example shows that the local declaration hides any global declaration: Consider the example below. Both loop counters have the same name. So, to reference the outer loop counter from the inner loop, you must use a label and dot notation, as follows: For example, the following loop normally executes ten times, but as soon as the FETCH statement fails to return a row, the loop completes no matter how many times it has executed: FOR j IN You can complete not only the current loop, but any enclosing loop. Occasionally, it can simplify logic enough to warrant its use. The NULL statement can improve readability by making the meaning and action of conditional statements clear. Overuse of GOTO statements can result in complex, unstructured code sometimes called spaghetti code that is hard to understand and maintain. So, use GOTO statements sparingly. For example, to branch from a deeply nested structure to an error-handling routine, raise an exception rather than use a GOTO statement. When executed, the GOTO statement transfers control to the labeled statement or block. In the following example, you go to an executable statement farther down in a sequence of statements: For example, the following GOTO statement is illegal: It can, however, improve readability. In a construct allowing alternative actions, the NULL statement serves as a placeholder. It tells readers that the associated alternative has not been overlooked, but that indeed no action is necessary. In the following example, the NULL statement shows that no action is taken for unnamed exceptions: The NULL statement is executable, so you can use it in clauses that correspond to circumstances in which no action is taken. In the following example, the NULL statement emphasizes that only top-rated employees get bonuses: A stub is dummy subprogram that allows you to defer the definition of a procedure or function until you test and debug the main program. In the following example, the NULL statement meets the requirement that at least one statement must appear in the executable part of a subprogram:

8: PSR Coding Style Guide - PHP-FIG

Simple control structures A program is usually not limited to a linear sequence of instructions. During its process it may bifurcate, repeat code or take decisions.

9: Simple control structures - C++ Tutorials

What Are the Three Types of Control Structures? The three basic types of control structures are sequential, selection and iteration. They can be combined in any way to solve a specified problem. Sequential is the default control structure, statements are executed line by line in the order in which.

CONTROL STRUCTURES pdf

Bridging generations: bringing the experiences of illness, health, and aging into the classroom Grace J. Youth with HIV/AIDS City of bones books The coming of a legend Pulse and digital circuits Hearing on split decision Molluscum Contagiosum 2008 pontiac g6 service manual Reminiscences of Georgia Laser ignition system in ic engine report Health, information, and migration Sodium status and the response to blockade of the renin-angiotensin system Friso L.H. Muntinghe and Gerja The Christmas Foundation Cumberland Co IL Marriages 1885-1893 My utmost his highest Youll never forget a name again! Diagnosis and Therapy of Porphyrias and Lead Intoxication State papers.bearing upon the purchase.of Louisiana. Irish Palatines in Ontario Discourses of Education in the Age of New Imperialism (Discourse, Power and Resistance Series) Tribal identity and minority status Select ing elementary Lessons in California history Supervised study; a discussion of the study lesson in high school The Boy Farmers of Elm Island Best friends think alike Chota bheem coloring book Adam Smiths moral philosophy Paleopathological diagnosis and interpretation A Darkness of the Soul Igcse chemistry formula sheet Wheatless cooking Jon schmidt christmas sheet music Priests of Moloch Letters of Franklin K. Lane, personal and political Sex Roles and the Ideal Society, by Richard Wasserstrom Action vs linking verbs Pneumothorax from gas-producing bacteria Whistler and Manet. Expressive and creative arts methods for trauma survivors