|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CS1400 Lab #8**  **Algorithm Design**  **Version 1.0**  **Objectives:**  At this point you should become familiar with ***Activity Diagrams*** and be very familiar with ***Pseudo-Code*** and the process of using them to design your program. After completing this lab, you should have used an ***Activity Diagrams*** and ***Pseudo-Code*** to analyze simple computing problems and be able to design ***algorithms*** for this program.  **Study Materials**  In order to do this exercise, you should go back through this week's set of slides on Program Design. Study the example shown in the slides thoroughly. Be sure that you understand the steps required to develop a program. A copy of the ***Algorithm Design Worksheet*** is located below. Use this worksheet to develop a solution to the programming exercise for this lab. As you work through the solution to the exercise, fill in the requested areas on the worksheet. This is not a time to be lazy! Good programmers spend a significant partition of their time making sure that they understand the problem, that they are trying to solve, and doing the hard design work. **Do not** try to write the program for this problem at this point and ***do not*** submit this ***Algorithm Worksheet***, it is to help you in designing the solution to this lab.  ***Algorithm Worksheet***  **Your Name: Spencer Carter Course and Section: CS 1400-03**  The purpose of this design worksheet is to help you organize your thoughts as you work out the solution to a particular programming problem. Filling out this worksheet is not busy work. Seasoned programmers know that it is much easier to write code once they have done the necessary design work. If you give your completed worksheet to someone else in the class, they should be able to write the program just from the information on this sheet.  **What is the problem to be solved?** In the space below, try to write in your own words a brief statement of the problem that is to be solved by this lab.    **What do I know?** In this space, write down any facts that you know about this problem.   1. What are the variables required? 2. What are the ***expressions*** (***equations***) required. Leave out extraneous information – stick to the facts that are required to solve the problem? Then write down any new facts that you have uncovered. If you found any ***mathematical relationships*** try to write down ***equations*** or ***formulas*** that express these relationships.   ***Remember, that the variables drive the design of your program.***   1. What user inputs are required? 2. What user outputs are required?   **What can I visualize about the problem?** In the space below sketch any pictures or ***Activity Diagrams*** that might help you visualize the problem to be solved. Label values, make notes of relationships, and look for patterns. Then write down any new facts that you have uncovered.  **What do I need from the user?** In this space, write down any other data that you need to collect from the user.  **What will my program produce?** In this space write down what it is that your program will produce. What outputs are required? Note any special formatting that may be required when you output this data.  **Line-by-line description of what the program needs to do using Pseudo-code:** Pseudo-Code is a list of English-like statements that precisely define the operations that your program will perform. In this space, write down line by line exactly what your program will do. Avoid using C# language constructs. Include all of the details that are necessary if someone were to write the program using your Pseudo-Code. **Test Values:** In the space below, write down at least three different possible sets of test values that you can use to test this program. These should include ***good values***, ***bad values*** and ***boundary values***. Using a calculator, spreadsheet or by hand compute the answers that you believe your program should produce. **Use these test values to verify your final program.** Remember, your program must work and ***NOT*** crash with any and all user inputs.  **Activity Diagram Programming Process**  The Activity Diagram Programming Process was developed to be used in the Unified Modelling Language (***UML***).  Activity Diagram  From Wikipedia, the free encyclopedia  [http://upload.wikimedia.org/wikipedia/commons/thumb/e/e7/Activity_conducting.svg/440px-Activity_conducting.svg.png](http://en.wikipedia.org/wiki/File:Activity_conducting.svg)  UML 1.x Activity Diagram for a guided [brainstorming](http://en.wikipedia.org/wiki/Brainstorming) [process](http://en.wikipedia.org/wiki/Mental_process)   |  | | --- | | [**UML diagrams**](http://en.wikipedia.org/wiki/Unified_Modeling_Language#Diagrams_overview) | | **Structural UML diagrams** | | * [Class diagram](http://en.wikipedia.org/wiki/Class_diagram) * [Component diagram](http://en.wikipedia.org/wiki/Component_diagram) * [Composite structure diagram](http://en.wikipedia.org/wiki/Composite_structure_diagram) * [Deployment diagram](http://en.wikipedia.org/wiki/Deployment_diagram) * [Object diagram](http://en.wikipedia.org/wiki/Object_diagram) * [Package diagram](http://en.wikipedia.org/wiki/Package_diagram) * [Profile diagram](http://en.wikipedia.org/wiki/Profile_diagram) | | **Behavioral UML diagrams** | | * **Activity diagram** * [Communication diagram](http://en.wikipedia.org/wiki/Communication_diagram) * [Interaction overview diagram](http://en.wikipedia.org/wiki/Interaction_overview_diagram) * [Sequence diagram](http://en.wikipedia.org/wiki/Sequence_diagram) * [State diagram](http://en.wikipedia.org/wiki/State_diagram_(UML)) * [Timing diagram](http://en.wikipedia.org/wiki/Timing_diagram_(Unified_Modeling_Language)) * [Use case diagram](http://en.wikipedia.org/wiki/Use_Case_Diagram) | | * [v](http://en.wikipedia.org/wiki/Template:UML_diagram_types) * [t](http://en.wikipedia.org/wiki/Template_talk:UML_diagram_types) * [e](http://en.wikipedia.org/w/index.php?title=Template:UML_diagram_types&action=edit) |   **Activity Diagrams** are graphical representations of [workflows](http://en.wikipedia.org/wiki/Workflow) of stepwise activities and actions[[1]](http://en.wikipedia.org/wiki/Activity_diagram#cite_note-1) with support for choice, iteration and concurrency. In the [Unified Modeling Language](http://en.wikipedia.org/wiki/Unified_Modeling_Language), Activity Diagrams are intended to model both computational and organizational processes (i.e. workflows).[[2]](http://en.wikipedia.org/wiki/Activity_diagram#cite_note-2)[[3]](http://en.wikipedia.org/wiki/Activity_diagram#cite_note-3) Activity Diagrams show the overall flow of control.  Activity Diagrams are constructed from a limited number of shapes, connected with arrows.[[4]](http://en.wikipedia.org/wiki/Activity_diagram#cite_note-4) The most important shape types:   * ***rounded rectangles*** represent ***actions***; **(just rectangles for CS1400)** * ***diamonds*** represent ***decisions***; * ***bars*** represent the ***start*** (*split*) or ***end*** (*join*) of concurrent activities; * ***black circle*** represents the **start** (**initial****state**) of the workflow; * ***encircled black circle*** represents the **end** (**final****state**).   ***Arrows*** run from the ***start*** towards the ***end*** and represent the order in which ***activities*** happen.  Activity Diagrams may be regarded as a form of [flowchart](http://en.wikipedia.org/wiki/Flowchart). Typical flowchart techniques lack constructs for expressing concurrency[[*citation needed*](http://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]. However, the ***join*** and ***split*** symbols in Activity Diagrams only resolve this for simple cases; the meaning of the model is not clear when they are arbitrarily combined with decisions or loops.  From the lab’s specification and the information from the ***Algorithm Design Worksheet*** draw the ***Activity Diagram*** and ***Pseudo-Code*** for this lab. Turn in the ***Activity Diagram*** with your ***Pseudo-Code*** in Project File.  **Pseudo-Code Programming Process**  The ***Pseudo-Code*** Programming Process was developed by Steve McConnell, and is explained in detail in his book ***"Code Complete*"** by Microsoft Press.  Before proceeding, you should have worked through the solution to the Farmer John problem using the Algorithm Design Worksheet. In this exercise, you will begin the process of writing the program for solving this problem. Carefully follow these steps:   * Create a Console Project, in Visual Studio, and name it Lab\_08. * From your Algorithm Design Worksheet copy the line by line description that you wrote for solving this problem in program’s ***Main*** ***method***. We call this line by line description **Pseudo-code**. It should accurately reflect the steps that your program will take, expressed in short English phrases. * Now paste this ***Pseudo-Code***, ***as comments***, into your program as shown below:   **// Project Prolog**  **// Name: Dennis Fairclough**  **// CS 1400 Section 002/003**  **// Project: Lab\_08**  **// Date: 1 Jan 2015**  **// Purpose: To illustrate the use of pseudo-code for program design.**  **//**  **// I declare that the following code was written by me or provided**  **// by the instructor for this project. I understand that copying source**  **// code from any other source constitutes cheating, and that I will receive**  **// a zero on this project if I am found in violation of this policy.**  **// ---------------------------------------------------------------------------**  **using** **System**;  **static** **class** Program  {  /// <summary>  /// Purpose: Pseudo-Code for Lab 08 in Main()  /// </summary>  **static** **void** **Main**()  {  **// Place Pseudo-Code here!**  Console**.Write**("\nPress Enter to continue ... ");  Console**.ReadLine**();  }**//End Main()**  }**//End class Program**   1. Place two forward slashes (single line comment operators) ***//*** in front of each line of ***Pseudo-Code***, so that they appear as comments in your Console Program. **Do not** try to write any other code at this point. Save your Project as you normally do.   **Programming Exercise**  Farmer John   |  | | --- | | **Programming Exercise: Problem Solving with the Algorithm design Worksheet**  **The problem**  circlesFarmer John has four fields of soybeans planted. There are three months left until harvest time, and the almanac has forecast a very dry summer. Farmer John has decided to put in a new, more efficient irrigation system to water his soybean crops. His estimate is that the new system will cost about $10,000. Since soybeans are going for about $12.75 a bushel, Farmer John wants to harvest as many bushels of soybeans as he possibly can. As he ponders on this problem, he sketches a map of his soybean fields that shows how the irrigation  system will cover his crops. This configuration is shown Fig. 1.  Figure Diagram of Farmer John's Field  The square ***ABCD*** has its vertices located at the centers of four identical circles, which are the regions covered by the new circular irrigation system. Farmer John is intrigued by the pattern shown in the shaded region of the map, and wonders what the area of this shaded region is. The problem you have to solve, is to compute the area of this shaded region (unwatered area).  **Deriving the Solution**  Follow the model outlined, in this week's slides, to develop an algorithm to solve this problem. Use the Algorithm Design Worksheet above, as a guide, filling out the appropriate sections as you go. These steps include gathering all of the important facts about the problem, analyzing the problem, deriving an equation for the unwatered shaded area, developing a step by step approach to solving the problem, drawing an **Activity Diagram and** writing **Pseudo-Code** and deriving the solution by hand so that you can test to see if your program works or is going to work. Look for variables, values, equations, expressions and methods that are needed.  Note that the only data that you need to solve this problem is the ***radius*** of one of the crop circles. You can calculate everything else that you need from this ***radius***, given the ***radius*** of one of the circles you can calculate the area of the circles and the area of the field. Let the user enter a value for the ***radius*** when the program runs. Note the units of measure that you are going to use for the ***radius***, i.e. inches, feet, yards, millimeters, meters, and kilometers, need to be stated in your ***Pseudo-Code***.  If you get stuck on this problem, ask about it in class. |   **File(s) to Submit:**  Zip your entire Project folder and rename it to **Lab\_08\_your-initials\_V1.0.zip**. For example, I would name my file **Lab\_08\_DAF\_V1.0.zip**. Submit this assignment as Lab\_08 on Canvas. Please only include your Activity Diagram, Pseudo-Code and Project folder in your zipped submission, do not include any other files in your submission.  There is an example executable file on Canvas for this lab.   |  |  |  | | --- | --- | --- | |  | **Grading Checklist** | C (correct)  X (incorrect) | | **#** | **Program** | **Submission** | | 1 | Meets & works to specifications | 6 points | | 2 | Error Free, elegant & efficient | 4 points | | 3 | Pseudo-Code | -3 points | | 4 | Style Guidelines | -2 points | | 6 | Source Files(s) & Formatting | -2 points | | 7 | Project Prolog | -1 points | | 8 | Function Prologs | -1 points | | 9 | Zip Filename | -1 points | | 10 | Lab & Project Names | -1 points | | 11 | Zip File is invalid or will not unzip | Lab = 0 pts | |  | Total Points | 10 | 0-9 | |