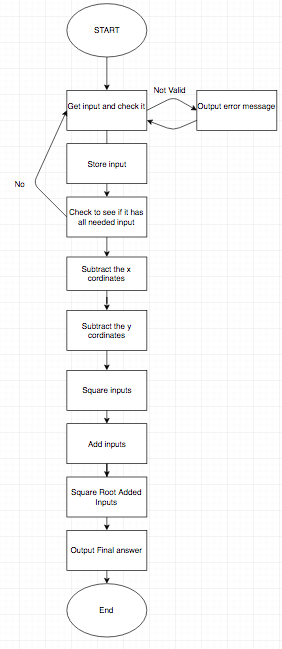
Distance Calculator Design Document

Utah Valley University

Caiden Kehrer, James Jenkins, Alex Jackman, Kaya Peter Kikudji, Dayson Hawkins

Architectural and Process Flow Diagram



Detailed Feature Descriptions and User Interface

This program will allow the user to input four values, two X and two Y. It contains a help feature in case the user is having difficulty understanding what they need to do to make the program function, or if they are curious about what the program is doing to reach its conclusions.   
The program will ensure that the values the user inputs are integers and in-bounds, otherwise sending an error message and requesting they try again. Once the user has given the program all four variables, it will use the Distance Formula to calculate the distance between the two points. The distance between the two will be printed to the console, and the user will be asked if they would like to exit or test more points. Choosing to exit closes the program, choosing to test more points begins the process from the beginning.

Data, Architectural

The program will open by informing the user of its intended purpose and asking if they would like further information or if they are ready to input their coordinates. Asking for more information will branch and show the user the equation for calculating distance as well as more details about inputs.

If the user instead inputs that they would like to continue, the program will branch further down and request the X coordinate of the first point, stating that it must be an integer. The input will be checked to ensure that it is useable data, containing no letters, decimal points, or other such unuseable information. If it is not, an error message is displayed and a coordinate is requested once again, with the prompt asking if the user would like to view the help page. Naturally this leads back to the information string with the prompt asking if they are ready to continue. This process is repeated for the first Y coordinate, and the full coordinates of the first point will be posted. The user will be asked to verify that this point is correct. If it is not, branch back to the start. If so, repeat the above processes for the second point.

Once the user has selected two valid points to be measured, the program will subtract the pairs of X and Y coordinates, then add the results to themselves on a loop of iterations equal to the starting numbers to square them. Should the end result turn up negative (tested by branch) it will be multiplied by negative one before proceeding. The two resulting numbers will be subtracted from each other, at long last leaving only a single number Final. As this number must be an integer (dividend and product of only integers) dividing it should be simple. Two floats are declared, A and B Using additive 2 loops Final will be divided by 2, the result placed in A. The dividend will be squared (into B) and compared with Final’s value. If it is smaller, the result of A plus B is divided by two into A and the loop repeats. If it is greater, A is divided by two and placed back into A, the loop repeating. If it is neither, branch to the exit. “Neither” is defined by subtraction of the Final and A equalling zero to five digits. If the result comes up as such, A is output to the console and declared as the distance between the two points. The user is asked if they would like to quit the program or start over, either leading to the HALT and END lines or branching back to the top.

Procedural Design

We have broken the steps needed to complete this assignment down and assigned them to various members of the development team. We will first tackle separate aspects of the square rooting loop and ensure that it is working properly as a top priority. User interface and input will be pushed to the back, with rotating constants used for initial testing to streamline the process. The division program is the second major priority, and will be worked on in conjunction with the root program during the in-depth design phase, as division is a crucial part of it. Once both of those steps have been completed and coded to function in vacuum, they will be connected and bug-tested. From there the square program should be a simple addition, alongside the subtraction and finally the implementation of user-input values. The final priority will be the information and restart branches, a simple but useful addition that shouldn’t take much time to add in. From there it will just be some simple bug-testing to completion.

Team Contributions

|  |  |  |
| --- | --- | --- |
| **Name** | **Primary Role** | **Contributions** |
| Dayson Hawkins | Developer | Developed Subroutines, Testing |
| Alex Jackman | Developer | Developed Subroutines, Testing |
| James Jenkins | SCRUM Master | Git Setup, UML Diagrams, Testing |
| Caiden Kehrer | Architect/Developer | Developed Code, Team Organizer, Testing |
| Kaya Peter Kikudji | Developer | Developed Code, Refactoring, Testing |

Code changes are as follows.

* Square function Caiden
* Division function Caiden
* Added prompt file Alex
* Modified the return on multiple files Dayson
* Added Square Root file Dayson
* Updated Addresses of each file Caiden
  + Division is now at x3000
  + Prompt is now at x3100
  + Square Root is now at x3200
  + Square is now at x3300
  + Program is now at x3400
  + These changes give each file roughly 300 lines of code to work with
* Program.asm now can run to getting #'s Caiden
* consolidate square root file Caiden
* Fixed Github error Caiden
* Detecting number of digits in user input Alex
* Now using loop to get input Caiden
* Recreate Division Caiden
* Output three digit numbers or greater Caiden
* Square root created Dayson
* Limit it to 120 as a max input Caiden

UML Diagrams of Subroutines

