Sep 17, 2018

Dong hee han

p457838

programming portfolio

activity 1.7

TABLE

[Data Structures 2](#_Toc525843527)

[Algorithms 2](#_Toc525843528)

[Pseudo code for plus, minus, divide and multiply 2](#_Toc525843529)

[Pseudo code for Sine, Cosine, Tangent 2](#_Toc525843530)

[Pseudo code for SquareRoot, CubeRT, Inverse 2](#_Toc525843531)

[Error handling techniques 3](#_Toc525843532)

[Recommended Testing Procedure 3](#_Toc525843533)

[Recommendations on upgrades and future enhancements 4](#_Toc525843534)

[1. Clarify upgrade processes 4](#_Toc525843535)

[2. Check Compatibility 4](#_Toc525843536)

[3. Plan, Test and Execute 4](#_Toc525843537)

[4. Notification of upgrade availability should be extensive to ensure that customers on maintenance are aware of the newest release 4](#_Toc525843538)

# Data Structures

|  |  |  |
| --- | --- | --- |
| Name | type | Purpose of each variables |
| total1 | double | Variables to save something from the calculation |
| total2 | double | Variables to save a final value to show on a screen |
| plusButtonClicked | Boolean | Variables to save an information that a user clicked a Plus button |
| minusButtonClicked | Boolean | Variables to save an information that a user clicked a Minus button |
| divideButtonClicked | Boolean | Variables to save an information that a user clicked a Divide button |
| multiplyButtonClicked | Boolean | Variables to save an information that a user clicked a Multiply button |
| sineButtonClicked | Boolean | Variables to save an information that a user clicked a Sine button |
| cosineButtonClicked | Boolean | Variables to save an information that a user clicked a Cosine button |
| tangentButtonClicked | Boolean | Variables to save an information that a user clicked a Tangent button |
| sqrtButtonClicked | Boolean | Variables to save an information that a user clicked a Sqrt button |
| cubeRButtonClicked | Boolean | Variables to save an information that a user clicked a cubeRT button |
| num | double | Temporary variables to save a double value |

# Algorithms

## Pseudo code for plus, minus, divide and multiply

if operator = "+" result = firstNumber + secondNumber

else if operator = "-" result = firstNumber - secondNumber

else if operator = "" result = firstNumber secondNumber

else if operator = "/" result = firstNumber / secondNumber

return result

## Pseudo code for Sine, Cosine, Tangent

if operator = "Sin" result = Math.sin ( Math.PI \* inputNumber / 180.0)

else if operator = "Cosine" result = Math.cosin ( Math.PI \* inputNumber / 180.0)

else if operator = "Tangent" = Math.tan ( inputNumber \* (Math.PI / 180.0))

return result

## Pseudo code for SquareRoot, CubeRT, Inverse

if operator = "SQRT" result = Math.Sqrt ( Math.PI \* inputNumber / 180.0)

else if operator = "cubeRT" result = Math.Pow ( inputNumber, (double)1 / 3)

else if operator = "Inverse" result = inverse operand(inputNumber)

return result

## Error handling techniques

* Validate an input value using String.IsNullOrEmpty(txtDisplay.Text)
* If there is no input value when a user click an operator, forced to set a zero
* Validate a number using a double.TryParse()
* If an input value is not a number, forced to return to the screen and show an error message
* Validate a valid number of the input value
* If an input value is 90 in a tangent calculation, forced to the screen and show an error message
* Validate a positive number
* If an input value is not a positive number in a Sine and Cosine calculation, forced to the screen and show an error message

# Recommended Testing Procedure

1. Set up the preparation for testing procedure

* Preparing a test case scenario for a White-box testing and black-box testing
* Make a schedule for each test and set a deadline
* Arranging testers to test a program

1. Run a White-box testing (each function test – Basic math, Trigonometric and Algebraic)

* Let developers test their function according to the test case scenario
* Write a test result and compare them to the expected results
* Find errors and fix it
* Retest and reconfirm it.

1. Run a unit and integration test

* Assign testers to run a test
* Let testers test a calculator according to a test case scenario.
* Write a test result and compare them to the expected results
* Pass the test result to the developers
* Find errors and fix it
* Make a test report

1. Arrange a meeting and make a review on the report

* Have a meeting to make a review on a testing result with team members
* Analyze it to find out whether the software implements the requirement
* Find out which area is weak and stable
* Let developers make up for their weakness according to the test result

1. Run a Black-box testing for a final release

* Let testers test a calculator without a test case scenario
* Write a test result and pass it to the test manager
* Analyze the test log
* Find out an anomalous events
* Rectify them

1. Installation and compatibility testing

* Set up a target environment to install a calculator software
* Test a compatibility with other application software and a range of operating systems

1. Wrap up and make a summary report

# Recommendations on upgrades and future enhancements

## 1. Clarify upgrade processes

Make the software upgrade process a PULL process, not a PUSH process

Customers should be able to get their upgrade when they are ready – not when the upgrade is made available.

Most customers prefer electronic software delivery over waiting to receive DVDs, inventorying them, etc

Provide customers a website where customers can find their maintenance coverage for all their products

Providing access to license entitlements is not enough anymore – customers must be able to view maintenance coverage for those licenses

## 2. Check Compatibility

Software compatibility is a critical component of the upgrade process. Therefore, before upgrading, the new version needs to be checked to ensure that it can operate with other peer and dependent software within the enterprise. If any software is not compatible or supported, a decision needs to be made if that software needs an upgrade of its own. It is not uncommon for IT to be managing several upgrades simultaneously to ensure compatibility between differing vendor’s software.

## 3. Plan, Test and Execute

Upgrading software can be complex and requires much process and planning; it needs to be treated and managed as a project. A project manager would be required to get business and technical buy in, plan the different activities from DEV installation, testing and cutover and execute the activities by having regular status checks. Resources would need to be assigned to do regression testing particularly those programs which touch other systems, which are critical to the business operation and which are complex in terms of logic and application.

4. Notification of upgrade availability should be extensive to ensure that customers on maintenance are aware of the newest releasePublishers want their customers to upgrade to the newest version (reduces support costs) so making sure that customers are aware of the new version is important