

Unidad Profesional Interdisciplinaria de Ingeniería y

Ciencias Sociales y Administrativas

Alumnos:

Gonzaléz Martinez Daniel

González Zarate Miguel Ángel

Sanchez Ramirez Brian Alexis

Fecha: 04/03/2024

Grupo: 5NV71

Materia: Ingeniería de Pruebas

Profesor: Cruz Martínez Ramón

Plan De Pruebas - Calculadora



1. **Análisis:**

Objetivo: Se busca realizar una calculadora la cual tenga las funciones básicas (suma, resta, multiplicación, división) además del apartado módulo y limpieza de pantalla, cumpliendo las pruebas de funcionalidad.

Misión: Nuestro programa realizará las funciones establecidas de manera correcta y cualquier usuario que la utilice no tendrá dificultades al utilizarla.

Visión: Se desea lograr un completo análisis para llevar un control del programa para prevenir problemáticas o bien en un plazo hacerle mejoras para un mejor trabajo.

Alcance: Este programa comenzará con las necesidades básicas que debe realizar una calculadora, se plantea hacerle mejoras en caso de ser necesarias en un plazo mediano de tiempo.

1. **Características del software:**

El lenguaje del programa será C# el cual es un lenguaje de programación orientado a objetos, creado por Microsoft para el desarrollo de aplicaciones en la plataforma .NET.

Algunas ventajas al utilizar este lenguaje son:

* Facilidad de aprendizaje.
* Seguridad: proporciona características que ayudan a proteger de vulnerabilidades.
* Permite realizar código rápido y reducción de errores y bugs.
* Es un lenguaje multiplataforma.
* Se cuenta con una comunidad la cual te puede ayudar en caso de dudas.
* Es un lenguaje de muy alta demanda, además de ser rentable provocando así un proyecto exitoso.

1. **Estrategia de Pruebas:**

Después de un análisis, se optó por el uso de los siguientes apoyos para una mejor administración:

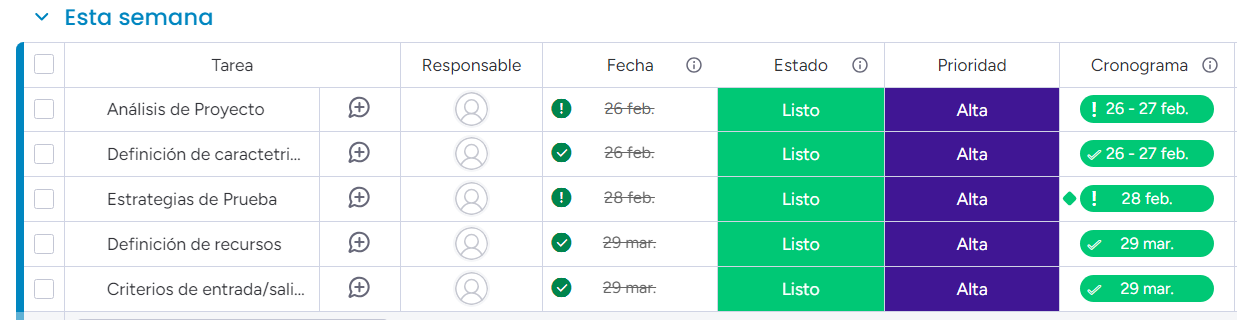
* Diagrama de flujo: Es una representación gráfica y secuencial de un proceso o flujo de trabajo con todas las tareas y actividades principales necesarias para lograr un objetivo común. Es de gran importancia destacar los procesos que deseamos realizar para ver si nuestro algoritmo tiene coherencia además que al hora de programar facilita mucho tener el diagrama de flujo de guía.
* Diagrama de casos de uso: Define una secuencia de acciones que da lugar a un resultado de valor observable. Por lo tanto, el uso de este diagrama nos permite visualizar aquellas acciones que pueden hacer los involucrados en el programa de la calculadora para un mejor comprendimiento de lo que se realiza al ejecutarse.

1. **Recursos Necesarios:**

El equipo se planteó primero las necesidades y requerimientos que deseamos que cumpla nuestro programa, teniendo en cuenta podemos continuar con la elaboración del diagrama de flujo el cual nos permitirá tener una mejor observación del algoritmo que se está elaborando además de visualizar el planteamiento que se está llevando.

Teniendo el diagrama de flujo realizado y comprendido por los integrantes, el diagrama de casos de uso nos permitirá asemejar aquellas acciones que se desean realizar con el programa y bien ver posibles dificultades que se pueden mejorar o bien, para una mejor comprensión del usuario destinado a utilizar el programa.

1. **Cronograma:**

****

1. **Criterios de entrada y salida:**

Se tendrán que plantear las variables de entrada las cuales fueron definidas de la siguiente manera:

1. private string:

* Num1 = string.Empty
* Num2 = string.Empty
* Simbolo=string.Empty
* operador=string.Empty
* simbolonum2=string.Empty

1. private int:

* i = 1
* j = 1
* k = 1

Para posteriormente continuar con el apartado de las operaciones básicas de nuestra calculadora las cuales son suma, resta, multiplicación y división, con la ayuda de un if dependiendo del botón que haya presionado el usuario se mostrará la operación predeterminada:

1. Suma: Se compara el operador con el símbolo “+” para continuar con la operación suma.

if (operador == "+")

{

txtResultado.Text = objet1.Suma(Num1,Num2);

Num1 = txtResultado.Text;

Num2 = string.Empty;

simbolo = string.Empty;

simbolonum2 = string.Empty;

operador = string.Empty;

}

1. Resta: Se compara el operador con el símbolo “-” para continuar con la operación resta.

if (operador == "-")

{

txtResultado.Text = objet1.Resta(Num1, Num2);

Num1 = txtResultado.Text;

Num2 = string.Empty;

simbolo = string.Empty;

simbolonum2 = string.Empty;

operador = string.Empty;

}

1. Multiplicación: Se compara el operador con el símbolo “ \* ” para continuar con la operación multiplicación.

if (operador == "\*")

{

txtResultado.Text = objet1.Multiplicacion(Num1, Num2);

Num1 = txtResultado.Text;

Num2 = string.Empty;

simbolo = string.Empty;

simbolonum2 = string.Empty;

operador = string.Empty;

}

1. División: Se compara el operador con el símbolo “/ ” para continuar con la operación división.

if (operador == "/")

{

txtResultado.Text = objet1.Division(Num1, Num2);

Num1 = txtResultado.Text;

Num2 = string.Empty;

simbolo = string.Empty;

simbolonum2 = string.Empty;

operador = string.Empty;

}

1. De igual manera, para el uso del modulo se estaria utilizando el simbolo “ % “ dandonos como resultado.

if (operador == "%")

{

txtResultado.Text = objet1.Modulo(Num1, Num2);

Num1 = txtResultado.Text;

Num2 = string.Empty;

simbolo = string.Empty;

simbolonum2 = string.Empty;

operador = string.Empty;

}

1. **Casos de prueba:**

| **Tipo de prueba** | **Descripción** | **Pasos** | **Resultado esperado** | **Estatus** |
| --- | --- | --- | --- | --- |
| Funcionalidad | Ingresar la expresión  1111111111111111111111111 + 2222222222222222222222222 | Ingresar un número mayor a 20 dígitos sumado a otro de 20 dígitos | Obtener como resultado  3333333333333333333333333 | **Pass** |
| Funcionalidad | Ingresar dos signos de suma | Ingresar la suma 2++2 | Efectuar la suma correctamente | **Fail** |
| Funcionalidad | Ingresar dos signos de menos | Ingresar la suma 2 - - 2 | Efectuar la multiplicación de signos correctamente | **Fail** |
| Funcionalidad | Ingresar dos signos de multiplicación | Ingresar la siguiente expresión 2 \*\* 2 | Provocar un error en la sintaxis | **Fail** |
| Funcionalidad | Ingresar dos signos de división | Ingresar la siguiente expresión 2 // 2 | Provocar un error en la sintaxis | **Fail** |
| Funcionalidad | Ingresar una operación que requiera jerarquía de operaciones | Ingresar la expresión 1 + 2 \* 3 | Obtener 7 como resultado | **Fail** |
| Funcionalidad | Probar que la calculadora sea multi argumento | Ingresar la expresión 1 + 2 + 3 + 4 + 5 - 6 | Obtener 9 como resultado | **Fail** |
| Funcionalidad | Probar que el resultado se guarde correctamente | Ingresar la expresión 1 + 2 posteriormente restar 4 al resultado | Obtener -1 como resultado | **Pass** |
| Funcionalidad | Probar que el programa no lance error por doble punto en un número | Ingresar la suma de 1..0 + 2..0 | Provocar error en la sintaxis | **Crash** |
| Funcionalidad | Probar que no acepte un numero con puntos adicionales | Ingresar …….1 | Provocar error de sintaxis o borrado de pantalla | **Pass** |
| Funcionalidad | Probar la notación científica | Ingresar 100000000000000000000 + 1 | 1E+20 | **Pass** |
| Funcionalidad | Probar la división entre 0 | Ingresar 1 / 0 | Error de sintaxis | **Pass** |
| Funcionalidad | Probar la división entre 0 y después restarle al error una unidad | Ingresar 1 / 0 y despues -1 | Error de sintaxis | **Fail** |
| Funcionalidad | Probar la división de 0 entre 0 | Ingresar 0 / 0 | Error de sintaxis | **Fail** |
| Funcionalidad | Probar falta de primer argumento en la suma | Ingresar + 2 | Error de sintaxis | **Fail** |
| Funcionalidad | Probar falta de primer argumento en la resta | Ingresar - 2 | Obtener -2 | **Pass** |
| Funcionalidad | Probar falta de primer argumento en la división | Ingresar / 2 | Error de sintaxis | **Fail** |
| Funcionalidad | Probar falta de primer argumento en la multiplicación | Ingresar \* 2 | Error de sintaxis | **Fail** |
| Funcionalidad | Probar falta de segundo argumento en la suma | Ingresar 2 + | Error de sintaxis | **Fail** |
| Funcionalidad | Probar falta de segundo argumento en la resta | Ingresar 2 - | Error de sintaxis | **Fail** |
| Funcionalidad | Probar falta de segundo argumento en la división | Ingresar 2 / | Error de sintaxis | **Fail** |
| Funcionalidad | Probar falta de segundo argumento en la multiplicación | Ingresar 2 \* | Error de sintaxis | **Fail** |
| Usabilidad | Probar la interfaz del usuario | Alargar los bordes de la calculadora | Interfaz de usuario más grande | **Fail** |

**Planificaciones de prueba**

Para planificar las pruebas anteriormente mencionadas, consultamos articulos y posts para observar cómo otras personas hicieron pruebas para determinado software o incluso pruebas para una calculadora como la que realizamos, obtuvimos mucha experiencia en realizar pruebas y ser más atentos a los detalles pequeños visitando páginas en todo internet, y a como realizar la tabla y con que elementos y como elaborar las descripciones y el status.

**Riesgos**

Probablemente estos errores de funcionalidad no representen ningún riesgo ya que no se le permite al usuario ingresar caracteres directamente del teclado, sin embargo, si es un riesgo desplegar la aplicación en el estado actual por que podría incomodar al usuario tal o cuales fallas en las que se llevaría una mala experiencia de nuestro software.

**Objetivos empresariales**

Dentro de los objetivos empresariales no tenemos uno mas en especifico para este software de calculadora, más que corregir todos los errores principalmente funcionales que encontramos para poder terminar exitosamente una calculadora y agregarla a nuestro portafolio de proyectos.

**Conclusiones**

Al realizar los casos de prueba pudimos observar que muchas de las pruebas no son pasadas por los tests, sin embargo, con el poco tiempo de desarrollo que tuvimos para realizar la aplicación pudimos darle las excepciones más comunes como lo es la división entre 0, además no la hicimos multi argumento por falta igualmente de tiempo por lo que hacerla para únicamente dos argumentos nos resulto mas comodo para cumplir con el tiempo de entrega. Hay muchos puntos en los que podemos mejorar nuestra calculadora y muchos tests en los que podemos mejorar para la próxima vez que nos encarguen programar una calculadora.

**Referencias bibliográficas:**

MYTASKPANEL (2020) Introducción a C#: Definición y utilidades. Recuperado desde: <https://www.mytaskpanel.com/introduccion-a-csharp/>

IBM (2021) Definición de casos de uso. Recuperado desde: <https://www.ibm.com/docs/es/engineering-lifecycle-management-suite/lifecycle-management/6.0.3?topic=requirements-defining-use-cases>

Asana (2024) ¿Qué es un diagrama de flujo y cómo hacerlo?. Recuperado desde: <https://asana.com/es/resources/what-is-a-flowchart>

Tes. (2024, 15 enero). Cómo escribir casos de prueba para software: ejemplos y tutorial. Parasoft. https://es.parasoft.com/blog/how-to-write-test-cases-for-software-examples-tutorial/

Estefafdez. (2020, 4 noviembre). 067- Pruebas Unitarias (IV): Ejemplo de la calculadora. Una QA En Apuros. https://unaqaenapuros.wordpress.com/2020/12/23/067-pruebas-unitarias-iv-ejemplo-de-la-calculadora/