./

Learning Report – Applied SDLC

Course Code: <CODE>



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared. By** | **Reviewed By** | **Approved By** | **Remarks/Revision Details** |
| 1.0 | 18-Sept-2020 | Anand A Warrier (99002602) |  |  | Draft report for review |
| 1.1 | 19-Sept-2020 | Anand A Warrier (99002602) |  |  | Updated with the report of mini project |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Document History**

# 

Contents

[Checklist 4](#_Toc51421280)

[Activity and Tasks 4](#_Toc51421281)

[Activity 1– System/Software Development 4](#_Toc51421282)

[Activity 2 – CI Workflow for C Programming 4](#_Toc51421283)

[Activity 3 – Agile Aspects 5](#_Toc51421284)

[Activity 1 – System/Software Development 5](#_Toc51421285)

[1. REQUIREMENTS 5](#_Toc51421286)

[2. DESIGN 7](#_Toc51421287)

[3. TEST PLAN 9](#_Toc51421288)

[Activity 2 - CI Workflow for C Programming 11](#_Toc51421289)

[1. GIT 11](#_Toc51421290)

[2. Make 14](#_Toc51421291)

[3. Build 14](#_Toc51421292)

[4. Code Quality 15](#_Toc51421293)

[Activity 3 – Agile Aspects 17](#_Toc51421294)

[1. Theme 17](#_Toc51421295)

[2. Epic 17](#_Toc51421296)

[3. User stories 17](#_Toc51421297)

[References 18](#_Toc51421298)

[Appendix 18](#_Toc51421299)

[Activity 4 – Project Report 20](#_Toc51421300)

[1. REQUIREMENTS 20](#_Toc51421301)

[2. DESIGN 21](#_Toc51421302)

[3. TEST PLAN 22](#_Toc51421303)

[Activity 5 - CI for Applied SDLC and Testing 23](#_Toc51421304)

[References 25](#_Toc51421305)

[Appendix 25](#_Toc51421306)

**List of Figures**

[Figure 1 - Deployment diagram 8](#_Toc51336820)

[Figure 2 - Component diagram 9](#_Toc51336821)

[Figure 3 - Use case diagram 9](#_Toc51336822)

[Figure 4 - Activity diagram 10](#_Toc51336823)

[Figure 5 - Overall GIT repository 12](#_Toc51336824)

[Figure 6 - Commits page 1 13](#_Toc51336825)

[Figure 7 - Commits page 2 14](#_Toc51336826)

[Figure 8 - Commits page 3 14](#_Toc51336827)

[Figure 9 - Commits page 4 15](#_Toc51336828)

[Figure 10 - Makefile 15](#_Toc51336829)

[Figure 11 - Build report 15](#_Toc51336830)

[Figure 12 - Codacy statistics 16](#_Toc51336831)

[Figure 13 - Issues reported by Codacy 16](#_Toc51336832)

[Figure 14 - cppcheck report 17](#_Toc51336833)

# Checklist

* Installation of SW on Phone and Desktop
* Professional Skills – Complete the 5 courses + POSH course
* Recap and Research on V and Agile for discussion on 17th Sept

# Activity and Tasks

## **Activity 1**– System/Software Development

* Section 1 – Requirements
  + Define/Select Topics
  + Research
    - Ageing.
    - Cost gradation
  + Define your System
  + SWOT Analysis
  + Detailed Requirements for your respective projects
    - High Level
    - Low level
* Section 2 – Design
  + High level Design /System Level
    - Structural Diagrams – 2
    - Behavioral Diagrams – 2
  + Low Level Design / SubSystem Level
    - Structural Diagrams – 2
    - Behavioral Diagrams – 2
* Section 3 – Test Plan
  + Requirement Based Test Plan
  + Scenario Based Test Plan
  + Boundary Condition based test plan

## **Activity 2** – CI Workflow for C Programming

* GIT
* Make
* Build
* Code Quality

## **Activity 3** – Agile Aspects

* Theme
* Epic
* User stories

# Activity 1 – System/Software Development

## **REQUIREMENTS**

Product identified: **HEADPHONES**

**Definition** [1]

• Headphones are a pair of small loudspeaker drivers worn on or around the head over a user's ears.

• They are electroacoustic transducers, which convert an electrical signal to a corresponding sound.

• They let a single user listen to an audio source privately, in contrast to a loudspeaker, which emits sound into the open air for anyone nearby to hear.

• Also known as ear speakers, earphones or, colloquially, cans.

**Research**

**Ageing:** [2]

• Single-sided headphones with heavy speakers (in 1890s)

• First headphone designed for personal listening (in 1930s)

• First stereo over the ear headphones (1950s)

• Sony Walkman headphones (in 1980s)

• Noise cancelling headphones by Bose (1990s)

• Apple iPod with in-ear headphones (2001)

• First true wireless in-ear headphones (2015)

• Apple AirPods were launched (2016)

**Cost gradation:** [3]

|  |  |  |
| --- | --- | --- |
| **On-ear** | **Over-the-ear** | **In-ear** |
| * Cost: Rs. 329 – 30,000 | * Cost: Rs. 1000 – 36,000 | * Cost: Rs. 299 – 20,000   (Rs. 21,000 for AirPods pro) |
| * Sits on the ear | * Completely covers the ear to avoid noise interference | * Extend into the ear canal and some come with clips to help them fit more securely on the ear. |

**Detailed requirements:**

|  |  |
| --- | --- |
| **High level requirements** | **Low level requirements** |
| * Faithfully reproduce the sound being played in the devices to the headphone speakers | * High sound quality |
| * Free of interference with external noise | * Comfortable to the ears |
| * Portable | * Aesthetically good-looking |
| * Low power consumption | * Mechanism to adjust the volume level from the headphone itself |
| * Durable | * Wire-less |
| * Lightweight | * Microphone feature |

**SWOT Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Strengths** | **Weaknesses** | **Opportunities** | **Threats** |
| * Very high demand in the market because of the fast-growing digital era | * Slow rate of innovation | * Room for price reduction | * Highly priced component to manufacture |
| * Easy to carry and portable | * Detachment from social interactions | * Room for innovation |  |
| * Encourages user privacy | * Prolonged use in high volume can cause ear damage |  |  |
| * Value for money |  |  |  |

## **DESIGN**

**Structural diagrams:**

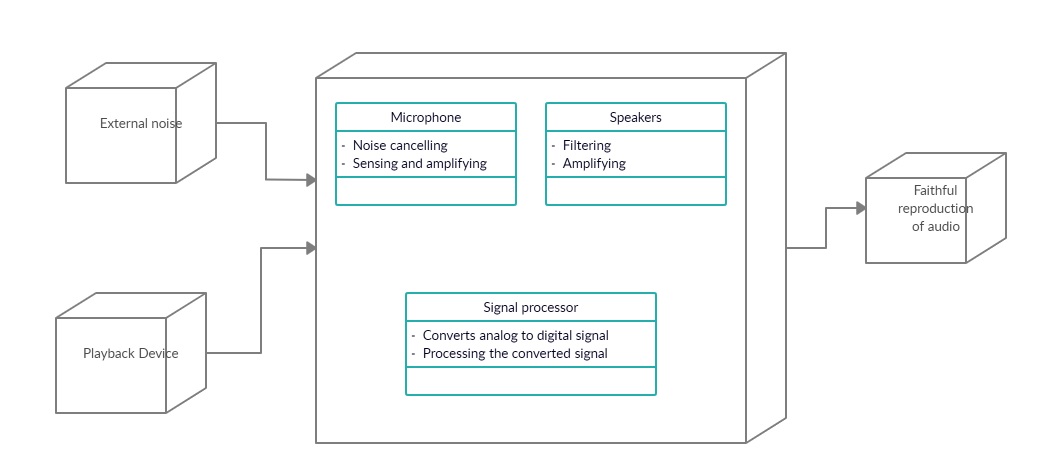


Figure - Deployment diagram

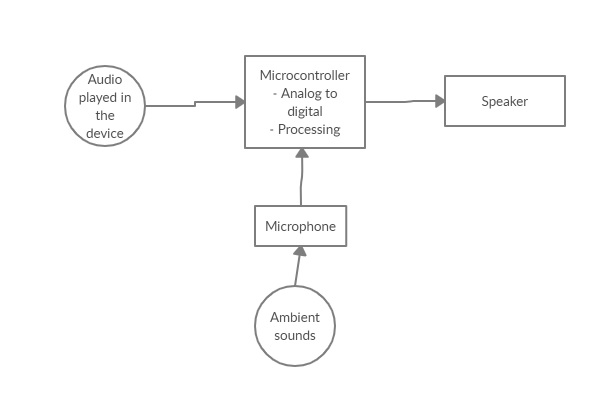


Figure - Component diagram

**Behavioral diagrams:**

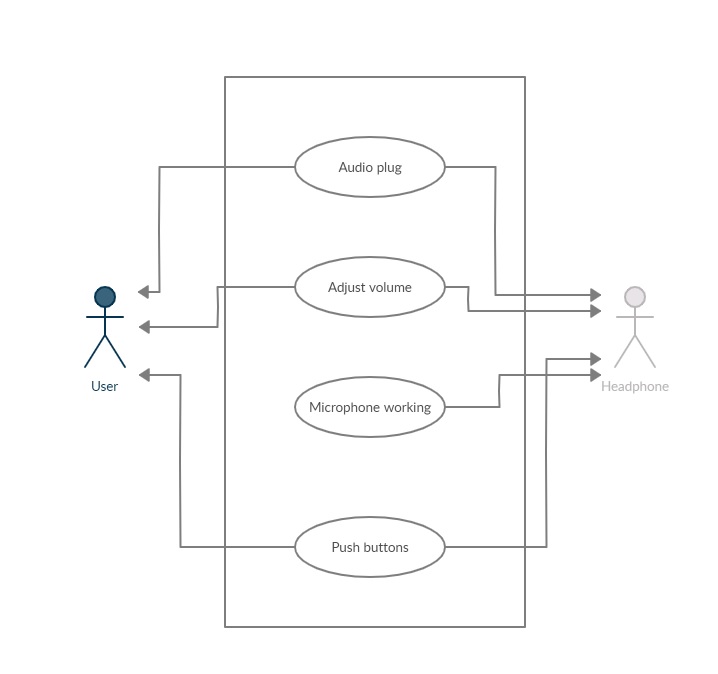
****

Figure - Use case diagram

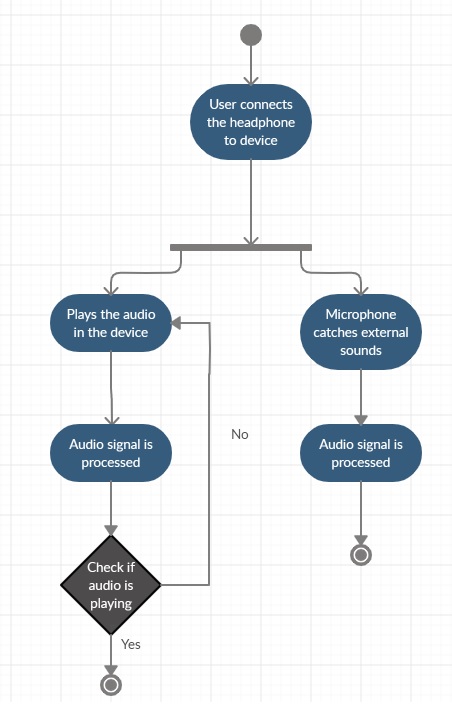
****

Figure - Activity diagram

## **TEST PLAN**

**Requirements gathering:**

|  |  |
| --- | --- |
| **ID** | **Description** |
| H\_01\_L\_01 | Faithfully reproduce the sound being played in the devices to the headphone speakers – High sound quality |
| H\_02\_L\_01 | Free of interference with external noise – High sound quality |
| H\_03\_H\_06 | Portable |
| H\_04\_H\_05 | Low power consumption |
| H\_05 | Durable |
| H\_06 | Lightweight |
| L\_02 | Comfortable to the ears |
| L\_03 | Aesthetically good-looking |
| L\_04 | Mechanism to adjust the volume level from the headphone itself |
| L\_05 | Wireless |
| L\_06 | Microphone feature |

**Test plan:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Description** | **Precondition** | **Expected input** | **Expected output** | **Actual output** |
| H\_01\_L\_01\_T01 | Faithfully reproduce the sound being played in the devices to the headphone speakers – High sound quality | Headphone must be connected to the device either through wire or wireless and device audio should be on | Any type of sound played in the device like audio player, keypad tones, ringtones and notification | Audible sound without any latency and any quality issues | Very slight latency in the order of milliseconds or microseconds. |
| H\_02\_L\_01\_T02 | Free of interference with external noise – High sound quality | Noise filter must be present inside the headphone | Any type of noise happening in the surroundings | Nothing in the surroundings is replicated in the speaker of the headphone | Very minimum noise |
| H\_03\_H\_06\_T03 | Portable - Lightweight | Minimal size for earpiece and wires | Place the headphone set inside a box of appropriate size | Sits perfectly inside the box | Sits inside the box |
| H\_04\_H\_05\_T04 | Low power consumption - Durable | Internal circuitry and components must be of good quality and low power consuming | A small voltage applied | Very low current for the same voltage measured by a multimeter | Low current for the same voltage measured by a multimeter |

# Activity 2 - CI Workflow for C Programming

## **GIT**

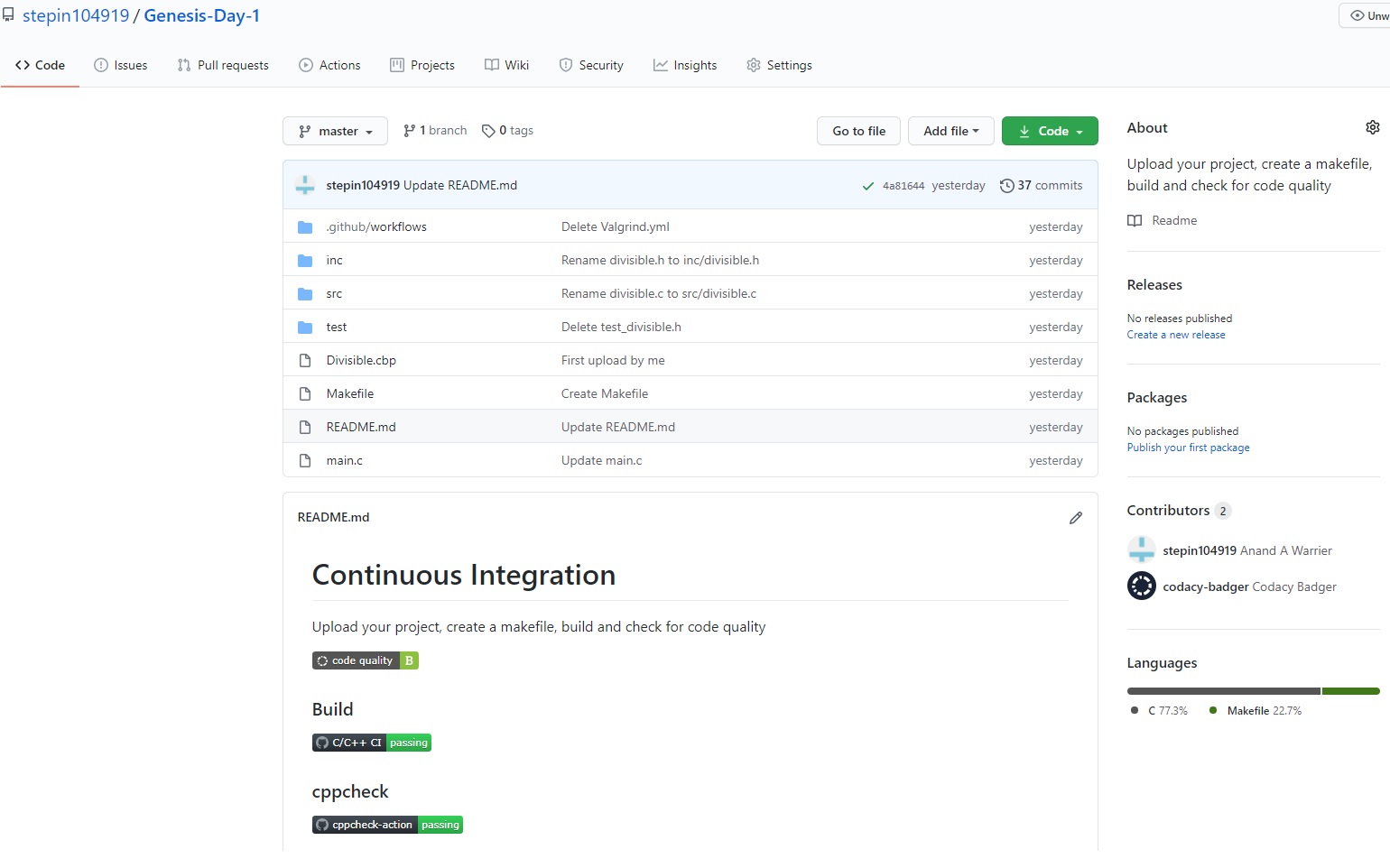


Figure - Overall GIT repository

<https://github.com/stepin104919/Genesis-Day-1>

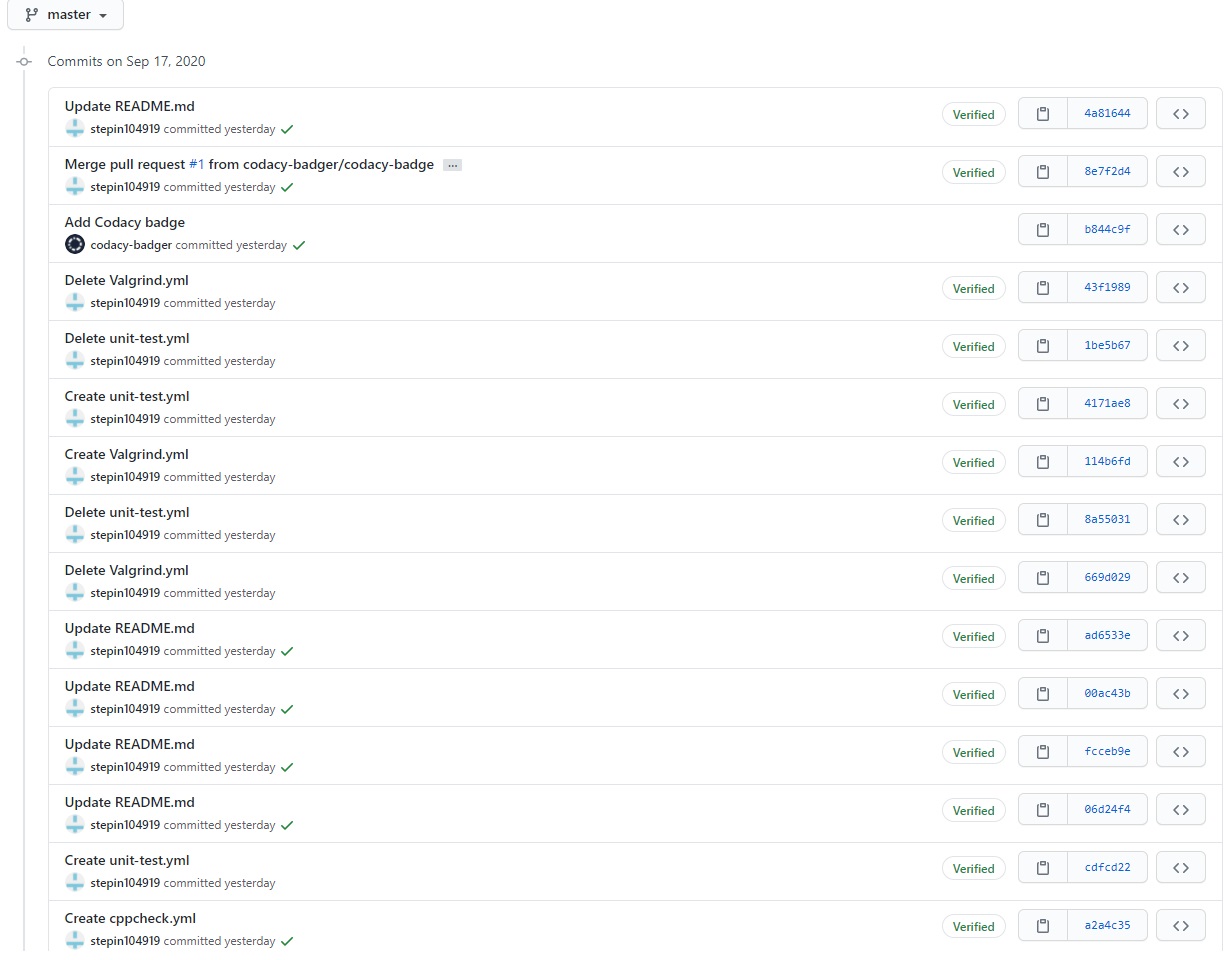


Figure - Commits page 1

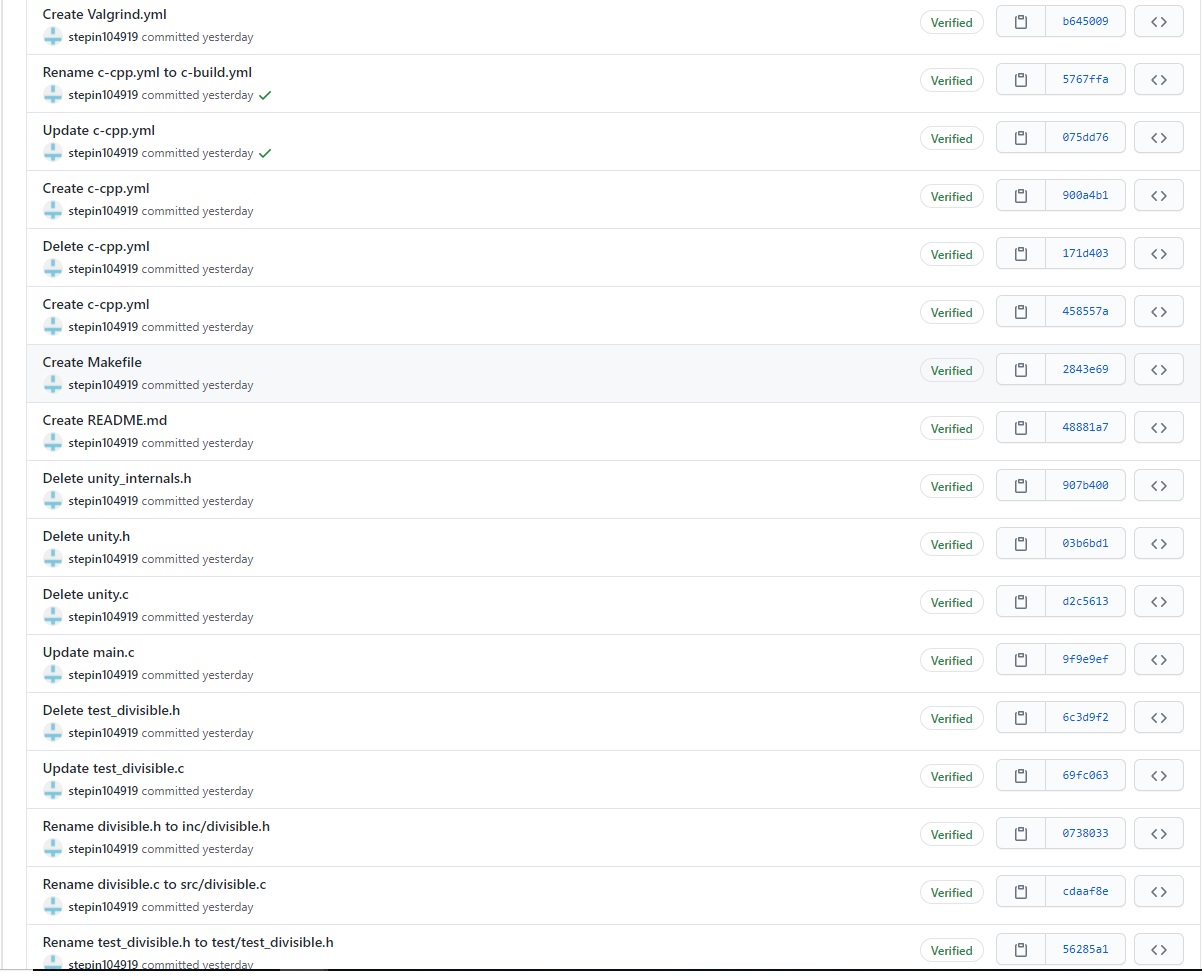


Figure - Commits page 2

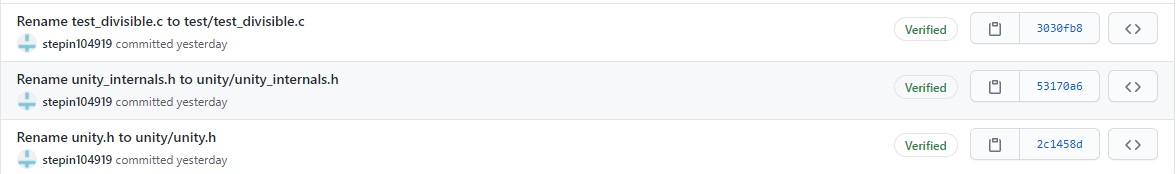
****

Figure - Commits page 3

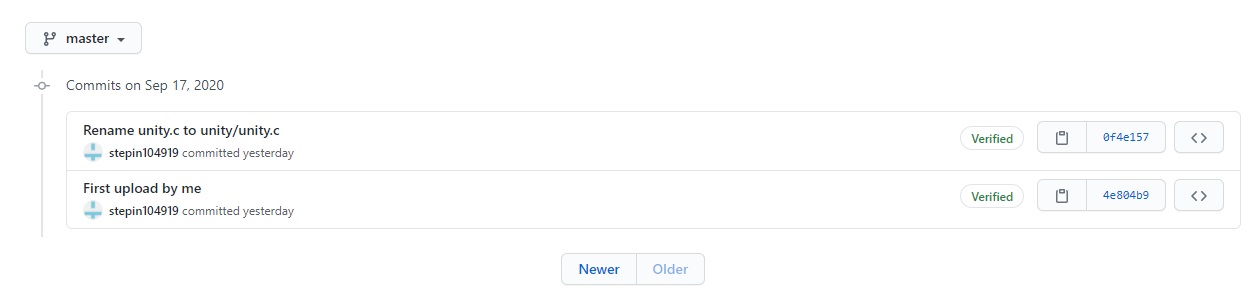
****

Figure - Commits page 4

## **Make**

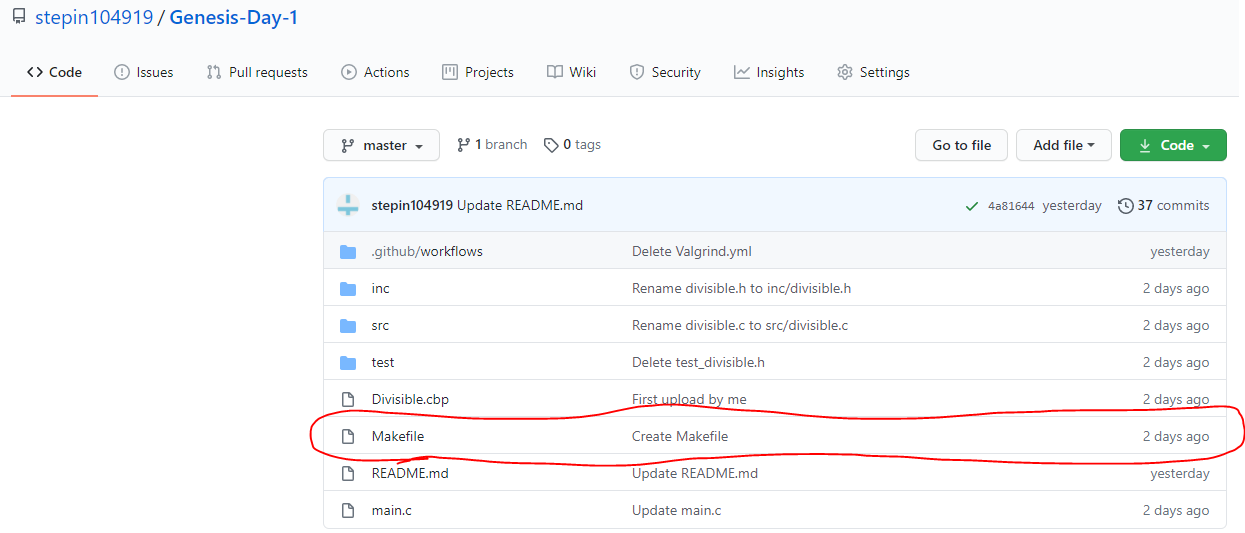


Figure - Makefile

## **Build**

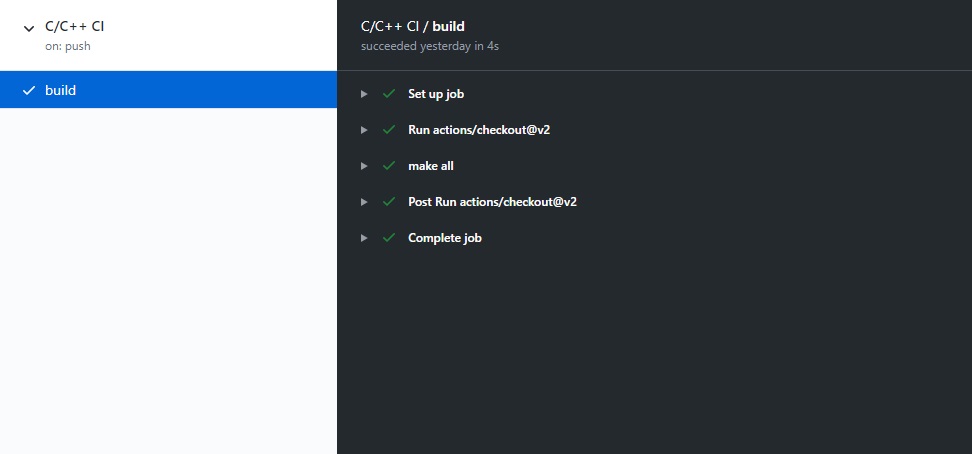


Figure - Build report

## **Code Quality**



Figure - Codacy statistics

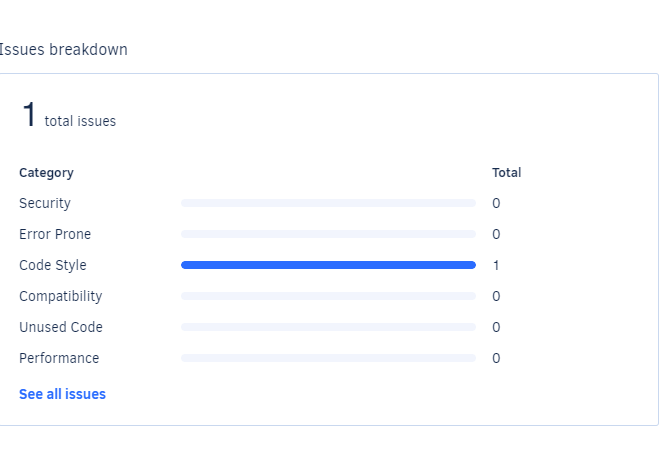


Figure - Issues reported by Codacy

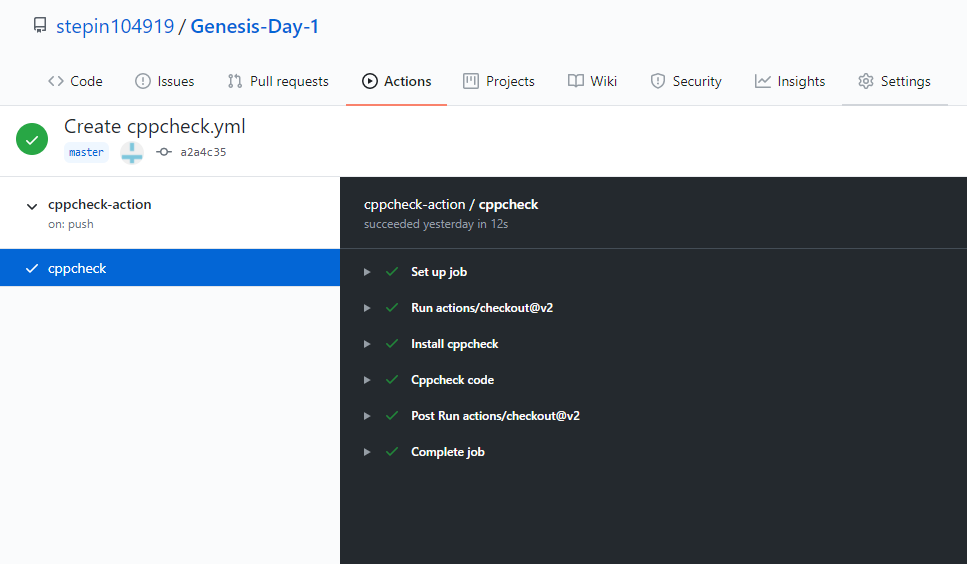


Figure - cppcheck report

# Activity 3 – Agile Aspects

## **Theme**

* Users want to listen to audio clips/music without disturbing others with the sound in public.
* A simple solution for this problem is required.
* Headphone/Earphone is the perfect product to do the job.
* With components of low cost, high quality headphones produced will be of great demand in the market.

## **Epic**

* End users include musicians, music producers, sound engineers and others.
* Type of headphones required vary according to customer’s tastes.
* Sound quality and noise rejection is a major factor, so the components have to be of high quality.

## **User stories**

* As a music enthusiast, I want to listen to music on the go privately.
* As an employee in a corporate firm, I very often need to attend in private video conferences during the day. I don’t want the meeting sound to be heard by others.
* As a sound engineer, I want to work with my projects effectively.

# References

[1] <https://en.wikipedia.org/wiki/Headphones>

[2] <https://www.headphonezone.in/blogs/audiophile-guide/evolution-of-headphones>

[3] <https://www.amazon.in>

# Appendix

**main.c**

|  |
| --- |
|  |
|  | #include<stdio.h>  #include <stdlib.h> |
|  | #include "divisible.h" |
|  |  |
|  | int main() |
|  | { |
|  | int num1,num2,flag; |
|  | printf("Enter dividend:\n"); |
|  | scanf("%d",&num1); |
|  | printf("Enter divisor:\n"); |
|  | scanf("%d",&num2); |
|  | flag=divisible(num1,num2); |
|  | switch(flag) |
|  | { |
|  | case -1: |
|  | printf("Division by zero not possible\n"); |
|  | break; |
|  | case 0: |
|  | printf("Not perfectly divisible\n"); |
|  | break; |
|  | default: |
|  | printf("Perfectly divisible\n"); |
|  | } |
|  | return 0; |
|  | } |

**divisible.c**

|  |
| --- |
|  |
|  | #include "divisible.h" |
|  | int divisible(int number1, int number2) |
|  | { |
|  | if(number2==0) |
|  | return -1; |
|  | else |
|  | { |
|  | if((number1%number2)==0) |
|  | return 1; |
|  | else |
|  | return 0; |
|  | } |
|  | } |

**divisible.h**

|  |
| --- |
|  |
|  | /\*\*  \* @file divisible.h |
|  | \* Program to check if the first number is perfectly divisible by second |
|  | \* |
|  | \*/ |
|  | #ifndef \_\_DIVISIBLE\_H\_\_ |
|  | #define \_\_DIVISIBLE\_H\_\_ |
|  |  |
|  | /\*\* |
|  | \* Check the divisibility of a number upon another number |
|  | \* @param[in] number which is to be divided |
|  | \* @param[in] number which is to be used to divide |
|  | \* @return 1 if perfectly divisible |
|  | \* @note Returns -1 if divisor is zero |
|  | \*/ |
|  | int divisible(int number1, int number2); |
|  |  |
|  | #endif /\* #ifndef \_\_FACTORIAL\_H\_\_ \*/ |

# Activity 4 – Project Report

Project name: **ADVANCED CALCULATOR**

**Problem statement**

Design, develop, test and run a calculator application with both simple and advanced features.

**Definition**

Calculator provides simple and advanced mathematical functions in an easy-to-use designed application.

* Perform basic calculations such as addition, subtraction, multiplication, and division.
* Do scientific operations such as logarithmic, factorial and exponential functions.

## **REQUIREMENTS**

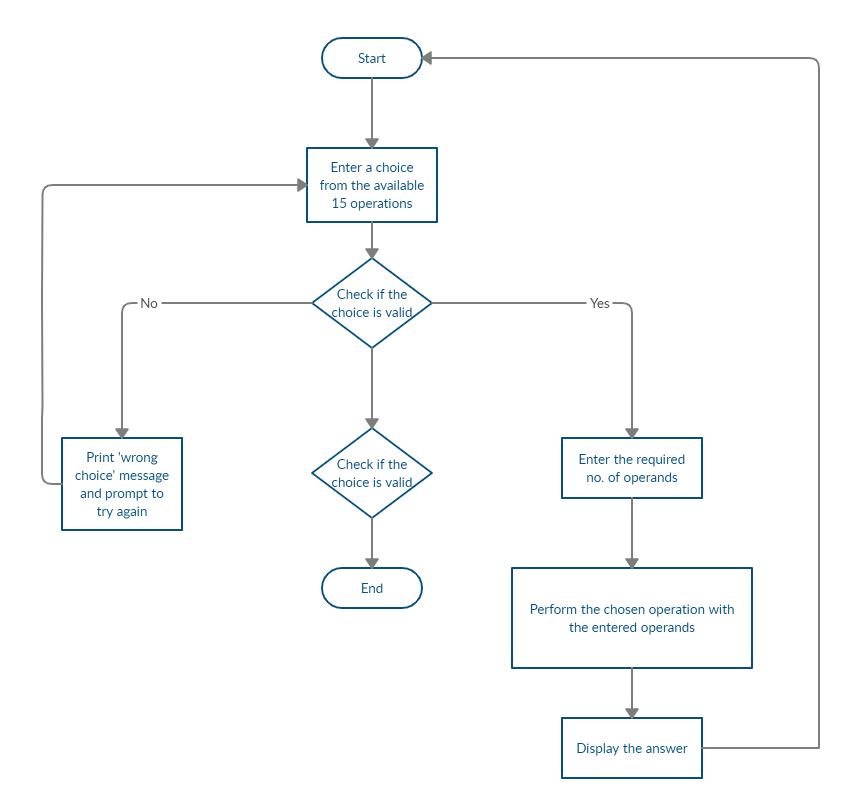
**High level requirements**

|  |  |
| --- | --- |
| **ID** | **Description** |
| H01 | Basic operations |
| H02 | Advanced operations |
| H03 | Report invalid operations |
| H04 | Report invalid inputs |

**Low level requirements**

|  |  |
| --- | --- |
| **ID** | **Description** |
| L01 | Codes for addition, subtraction, multiplication and division |
| L02 | Codes for modulo, factorial, power, antilogarithm, common logarithm, natural logarithm, absolute value, inverse, square root, square & cube |
| L03 | Codes to display a message reporting invalid operation |
| L04 | Codes to reject unexpected inputs and try again |

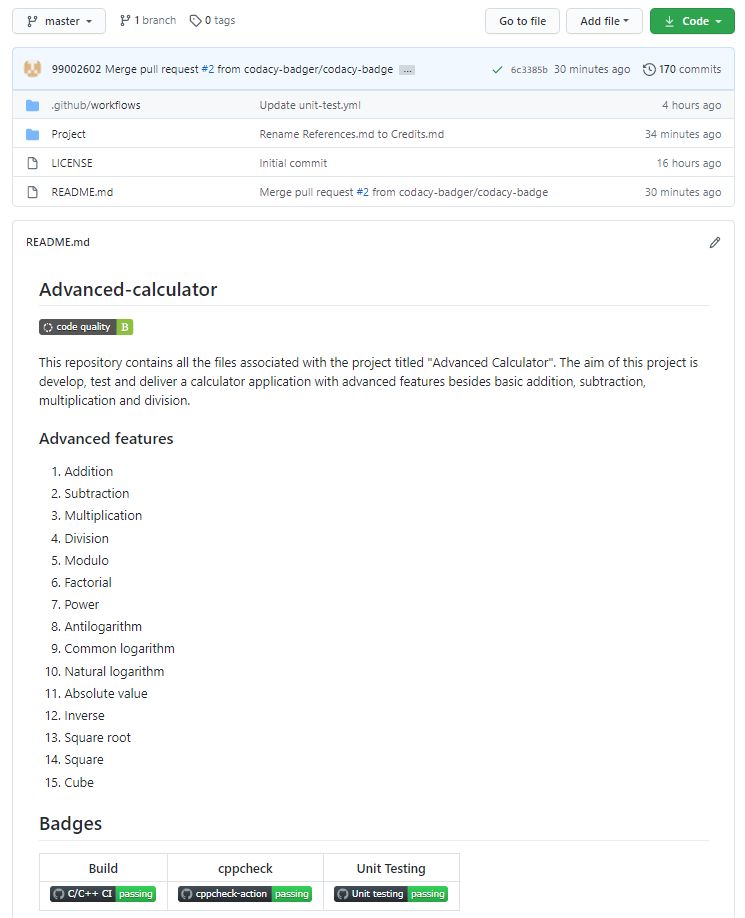
## **DESIGN**



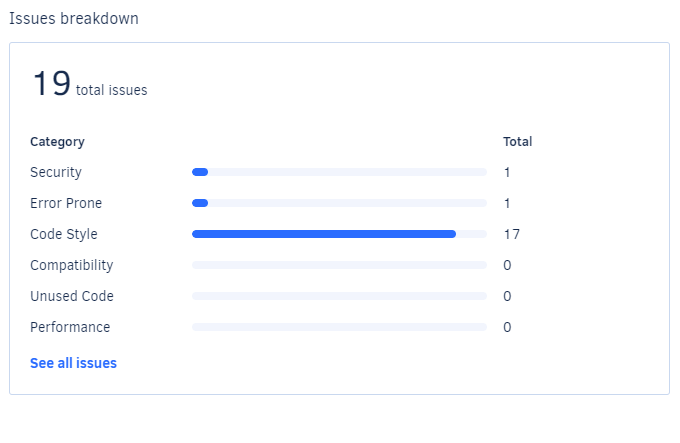
## **TEST PLAN**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | Description | Precondition | Expected input | Expected output | Actual output |
| H01 | Basic operations | Operation number entered must be between 1 and 4 | 4 | Division | Division |
| H02 | Advanced operations | Operation number entered must be between 5 and 15 | 9 | Common Logarithm | Common Logarithm |
| H03 | Report invalid operations | Operation number entered must not be between 1 and 16 | 20 | Invalid operation | Invalid operation |
| H04 | Report invalid inputs | Size of operand(s) entered exceeds the size of type of operand(s) | 3.5e+38 | 3.5e+38 | Size exceeded |
| H04 | Report invalid inputs | Size of result of an operation exceeds the size for which it is defined | factorial(20) | 2.4e+18 | Size exceeded |
| H04 | Report invalid inputs | Operand entered for division/modulo/inverse is zero | inverse(0) | Not defined | Error. Try again |

# Activity 5 - CI for Applied SDLC and Testing



<https://github.com/99002602/Advanced-calculator>





# References

[1] <https://github.com/stepin654321/MiniProject_Template/tree/master/MiniProject_C/3_Implementation>

# Appendix

**main.c**

#include<math.h>

|  |
| --- |
|  |
|  | #include <advanced\_calculator.h> |
|  |  |
|  | /\* Status of the operation requested \*/ |
|  | #define VALID (1) |
|  | #define INVALID (0) |
|  |  |
|  | /\* Calculator operation requested by user\*/ |
|  | unsigned int calculator\_operation = 0; |
|  |  |
|  | /\* Operands on which calculation is performed \*/ |
|  | float calculator\_operand1 = 0; |
|  | float calculator\_operand2 = 0; |
|  | int calculator\_operand3 = 0; |
|  | int calculator\_operand4 = 0; |
|  |  |
|  | /\* Valid operations \*/ |
|  | enum operations{ ADD=1, SUBTRACT, MULTIPLY, DIVIDE, MODULO, FACTORIAL, POWER, ANTILOG, COMMONLOG, NATURALLOG, MODULUS, INVERSE, ROOT, SQUARE, CUBE, EXIT }; |
|  |  |
|  | /\* Display the menu of operations supported \*/ |
|  | void calculator\_menu(void); |
|  | /\* Verifies the requested operations validity \*/ |
|  | int valid\_operation(int operation); |
|  |  |
|  |  |
|  | /\* Start of the application \*/ |
|  | int main(int argc, char \*argv[]) |
|  | { |
|  | printf("\n\*\*\*\*Calculator\*\*\*\*\n"); |
|  | while(1) |
|  | { |
|  | calculator\_menu(); |
|  | } |
|  | } |
|  |  |
|  | void calculator\_menu(void) |
|  | { |
|  | printf("\nAvailable Operations\n"); |
|  | printf("\n1. ADD\n2. SUBTRACT\n3. MULTIPLY\n4. DIVIDE\n5. MODULO\n6. FACTORIAL\n7. POWER\n8. ANTILOG\n9. COMMONLOG\n10. NATURALLOG\n11. MODULUS\n12. INVERSE\n13. ROOT\n14. SQUARE\n15. CUBE\n16. Exit"); |
|  | printf("\n\tEnter your choice\n"); |
|  |  |
|  | \_\_fpurge(stdin); |
|  | scanf("%u", &calculator\_operation); |
|  |  |
|  | if(EXIT == calculator\_operation) |
|  | { |
|  | printf("\nThank you. Exiting the Application\n"); |
|  | exit(0); |
|  | } |
|  |  |
|  | if(INVALID != valid\_operation(calculator\_operation)) |
|  | { |
|  | printf("\n\tEnter your Numbers with space between them\n"); |
|  | \_\_fpurge(stdin); |
|  | if((calculator\_operation>=1 && calculator\_operation<=4) || (calculator\_operation==7)) |
|  | scanf("%f %f", &calculator\_operand1, &calculator\_operand2); |
|  | else if(calculator\_operation>=8) |
|  | scanf("%f",&calculator\_operand1); |
|  | else if(calculator\_operation==5) |
|  | scanf("%d %d",&calculator\_operand3,&calculator\_operand4); |
|  | else if(calculator\_operation==6) |
|  | scanf("%d",&calculator\_operand3); |
|  | } |
|  | else |
|  | { |
|  | printf("\n\t---Wrong choice---\nEnter to continue\n"); |
|  | \_\_fpurge(stdin); |
|  | getchar(); |
|  | return; |
|  |  |
|  | } |
|  | switch(calculator\_operation) |
|  | { |
|  | case ADD: |
|  | printf("\n\t%f + %f = %f\nEnter to continue", |
|  | calculator\_operand1, |
|  | calculator\_operand2, |
|  | add(calculator\_operand1, calculator\_operand2)); |
|  |  |
|  | \_\_fpurge(stdin); |
|  | getchar(); |
|  | break; |
|  | case SUBTRACT: |
|  | printf("\n\t%f - %f = %f\nEnter to continue", |
|  | calculator\_operand1, |
|  | calculator\_operand2, |
|  | subtract(calculator\_operand1, calculator\_operand2)); |
|  |  |
|  | \_\_fpurge(stdin); |
|  | getchar(); |
|  | break; |
|  | case MULTIPLY: |
|  | printf("\n\t%f \* %f = %f\nEnter to continue", |
|  | calculator\_operand1, |
|  | calculator\_operand2, |
|  | multiply(calculator\_operand1, calculator\_operand2)); |
|  |  |
|  | \_\_fpurge(stdin); |
|  | getchar(); |
|  | break; |
|  | case DIVIDE: |
|  | printf("\n\t%f / %f = %f\nEnter to continue", |
|  | calculator\_operand1, |
|  | calculator\_operand2, |
|  | divide(calculator\_operand1, calculator\_operand2)); |
|  |  |
|  | \_\_fpurge(stdin); |
|  | getchar(); |
|  | break; |
|  | case MODULO: |
|  | printf("\n\t%d %% %d = %d\nEnter to continue", |
|  | calculator\_operand3, |
|  | calculator\_operand4, |
|  | modulo(calculator\_operand3, calculator\_operand4)); |
|  |  |
|  | \_\_fpurge(stdin); |
|  | getchar(); |
|  | break; |
|  | case FACTORIAL: |
|  | printf("\n\t%d! = %d\nEnter to continue", |
|  | calculator\_operand3, |
|  | factorial(calculator\_operand3)); |
|  |  |
|  | \_\_fpurge(stdin); |
|  | getchar(); |
|  | break; |
|  | case POWER: |
|  | printf("\n\t%f ^ %f = %f\nEnter to continue", |
|  | calculator\_operand1, |
|  | calculator\_operand2, |
|  | power(calculator\_operand1, calculator\_operand2)); |
|  |  |
|  | \_\_fpurge(stdin); |
|  | getchar(); |
|  | break; |
|  | case ANTILOG: |
|  | printf("\n\tantilog(%f) = %f\nEnter to continue", |
|  | calculator\_operand1, |
|  | antilog(calculator\_operand1)); |
|  |  |
|  | \_\_fpurge(stdin); |
|  | getchar(); |
|  | break; |
|  | case COMMONLOG: |
|  | printf("\n\tlog(%f) = %f\nEnter to continue", |
|  | calculator\_operand1, |
|  | base10log(calculator\_operand1)); |
|  |  |
|  | \_\_fpurge(stdin); |
|  | getchar(); |
|  | break; |
|  | case NATURALLOG: |
|  | printf("\n\tln(%f) = %f\nEnter to continue", |
|  | calculator\_operand1, |
|  | naturallog(calculator\_operand1)); |
|  |  |
|  | \_\_fpurge(stdin); |
|  | getchar(); |
|  | break; |
|  | case MODULUS: |
|  | printf("\n\t|%f| = %f\nEnter to continue", |
|  | calculator\_operand1, |
|  | modulus(calculator\_operand1)); |
|  |  |
|  | \_\_fpurge(stdin); |
|  | getchar(); |
|  | break; |
|  | case INVERSE: |
|  | printf("\n\t%f ^ -1 = %f\nEnter to continue", |
|  | calculator\_operand1, |
|  | inverse(calculator\_operand1)); |
|  |  |
|  | \_\_fpurge(stdin); |
|  | getchar(); |
|  | break; |
|  | case ROOT: |
|  | printf("\n\tSquare Root of %f = %f\nEnter to continue", |
|  | calculator\_operand1, |
|  | root(calculator\_operand1)); |
|  |  |
|  | \_\_fpurge(stdin); |
|  | getchar(); |
|  | break; |
|  | case SQUARE: |
|  | printf("\n\t%f ^ 2 = %f\nEnter to continue", |
|  | calculator\_operand1, |
|  | square(calculator\_operand1)); |
|  |  |
|  | \_\_fpurge(stdin); |
|  | getchar(); |
|  | break; |
|  | case CUBE: |
|  | printf("\n\t%f ^ 3 = %f\nEnter to continue", |
|  | calculator\_operand1, |
|  | cube(calculator\_operand1)); |
|  |  |
|  | \_\_fpurge(stdin); |
|  | getchar(); |
|  | break; |
|  | case 16: |
|  | exit(0); |
|  | break; |
|  | default: |
|  | printf("\n\t---It should never come here---\n"); |
|  | } |
|  | } |
|  |  |
|  | int valid\_operation(int operation) |
|  | { |
|  | /\* Check if the operation is a valid operation \*/ |
|  | return ((ADD <= operation) && (EXIT >= operation)) ? VALID: INVALID; |
|  | } |

**advanced\_calculator.c**

|  |
| --- |
|  |
|  | #include<math.h>  #include <advanced\_calculator.h> |
|  |  |
|  | float add(float operand1, float operand2) |
|  | { |
|  | return operand1 + operand2; |
|  | } |
|  |  |
|  | float subtract(float operand1, float operand2) |
|  | { |
|  | return operand1 - operand2; |
|  | } |
|  |  |
|  | float multiply(float operand1, float operand2) |
|  | { |
|  | return operand1 \* operand2; |
|  | } |
|  |  |
|  | float divide(float operand1, float operand2) |
|  | { |
|  | if(0 == operand2) |
|  | return 0; |
|  | else |
|  | return operand1 / operand2; |
|  | } |
|  |  |
|  | int modulo(int operand1, int operand2) |
|  | { |
|  | if(operand1==0) |
|  | return 0; |
|  | else |
|  | return operand1%operand2; |
|  | } |
|  |  |
|  | int factorial(int operand1) |
|  | { |
|  | if(operand1<=0) |
|  | return 1; |
|  | else |
|  | return operand1\*factorial(operand1-1); |
|  | } |
|  |  |
|  | float power(float operand1, float operand2) |
|  | { |
|  | return pow(operand1,operand2); |
|  | } |
|  |  |
|  | float antilog(float operand1) |
|  | { |
|  | return pow(10,operand1); |
|  | } |
|  |  |
|  | float base10log(float operand1) |
|  | { |
|  | if(operand1==0) |
|  | return 0; |
|  | else |
|  | return log10(operand1); |
|  | } |
|  |  |
|  | float naturallog(float operand1) |
|  | { |
|  | if(operand1==0) |
|  | return 0; |
|  | else |
|  | return log(operand1); |
|  | } |
|  |  |
|  | float modulus(float operand1) |
|  | { |
|  | return (operand1>=0)?operand1:(-operand1); |
|  | } |
|  |  |
|  | float inverse(float operand1) |
|  | { |
|  | if(operand1==0) |
|  | return 0; |
|  | else |
|  | return 1/operand1; |
|  | } |
|  |  |
|  | float root(float operand1) |
|  | { |
|  | if(operand1<0) |
|  | return 0; |
|  | else |
|  | return sqrt(operand1); |
|  | } |
|  |  |
|  | float square(float operand1) |
|  | { |
|  | return operand1\*operand1; |
|  | } |
|  |  |
|  | float cube(float operand1) |
|  | { |
|  | return operand1\*operand1\*operand1; |
|  | } |

**advanced\_calculator.h**

|  |
| --- |
|  |
|  | /\*\*  \* @file advanced\_calculator.h |
|  | \* Calculator application with 15 mathematical operations |
|  | \* |
|  | \*/ |
|  | #ifndef \_\_ADVANCED\_CALCULATOR\_H\_\_ |
|  | #define \_\_ADVANCED\_CALCULATOR\_H\_\_ |
|  |  |
|  | #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <stdio\_ext.h> |
|  | #include <math.h> |
|  | /\*\* |
|  | \* adds the operand1 and operand2 and returns the result |
|  | \* @param[in] operand1 |
|  | \* @param[in] operand2 |
|  | \* @return Result of operand1 + operand2 |
|  | \*/ |
|  | float add(float operand1, float operand2); |
|  |  |
|  | /\*\* |
|  | \* subtracts the operand1, operand2 and returns the result |
|  | \* @param[in] operand1 |
|  | \* @param[in] operand2 |
|  | \* @return Result of operand1 - operand2 |
|  | \*/ |
|  | float subtract(float operand1, float operand2); |
|  |  |
|  | /\*\* |
|  | \* multiply the operand1, operand2 and returns the result |
|  | \* @param[in] operand1 |
|  | \* @param[in] operand2 |
|  | \* @return Result of operand1 \* operand2 |
|  | \*/ |
|  | float multiply(float operand1, float operand2); |
|  |  |
|  |  |
|  | /\*\* |
|  | \* divides the operand1 by operand2 and returns the result |
|  | \* @param[in] operand1 |
|  | \* @param[in] operand2 |
|  | \* @return floating point value of the operand1 / operand2 |
|  | \* @note returns 0 for divide by 0 error |
|  | \*/ |
|  | float divide(float operand1, float operand2); |
|  |  |
|  |  |
|  | /\*\* |
|  | \* divides the operand1 by operand2 and returns the remainder |
|  | \* @param[in] operand1 |
|  | \* @param[in] operand2 |
|  | \* @return integer value of the operand1 / operand2 |
|  | \* @note returns 0 for divide by 0 error |
|  | \*/ |
|  | int modulo(int operand1,int operand2); |
|  |  |
|  |  |
|  | /\*\* |
|  | \* finds the factorial of operand1 and returns the result |
|  | \* @param[in] operand1 |
|  | \* @return integer value of the operand1 factorial |
|  | \* @note returns 1 for factorial of 0 and negative integers |
|  | \*/ |
|  | int factorial(int operand1); |
|  |  |
|  |  |
|  | /\*\* |
|  | \* raises the operand1 to the power operand2 and returns the result |
|  | \* @param[in] operand1 |
|  | \* @param[in] operand2 |
|  | \* @return floating point value of operand1 to the power operand2 |
|  | \*/ |
|  | float power(float operand1, float operand2); |
|  |  |
|  |  |
|  | /\*\* |
|  | \* finds the antilogarithm of operand1 to the base 10 and returns the result |
|  | \* @param[in] operand1 |
|  | \* @return floating point value of 10 to the power operand1 |
|  | \*/ |
|  | float antilog(float operand1); |
|  |  |
|  |  |
|  | /\*\* |
|  | \* finds logarithm of operand1 to the base 10 and returns the result |
|  | \* @param[in] operand1 |
|  | \* @return floating point value of logarithm of operand1 to the base 10 |
|  | \* @note returns 0 as invalid input |
|  | \*/ |
|  | float base10log(float operand1); |
|  |  |
|  |  |
|  | /\*\* |
|  | \* finds natural logarithm of operand1 and returns the result |
|  | \* @param[in] operand1 |
|  | \* @return floating point value of natural logarithm of operand1 |
|  | \* @note returns 0 as invalid input |
|  | \*/ |
|  | float naturallog(float operand1); |
|  |  |
|  |  |
|  | /\*\* |
|  | \* finds absolute value of operand1 and returns the result |
|  | \* @param[in] operand1 |
|  | \* @return floating point value of absolute value of operand1 |
|  | \*/ |
|  | float modulus(float operand1); |
|  |  |
|  |  |
|  | /\*\* |
|  | \* finds the reciprocal of operand1 and returns the result |
|  | \* @param[in] operand1 |
|  | \* @return floating point value of reciprocal of operand1 |
|  | \* @note returns 0 for divide by 0 error |
|  | \*/ |
|  | float inverse(float operand1); |
|  |  |
|  |  |
|  | /\*\* |
|  | \* finds square root of operand1 and returns the result |
|  | \* @param[in] operand1 |
|  | \* @return floating point value of square root of operand1 |
|  | \* @note returns 0 for negative numbers |
|  | \*/ |
|  | float root(float operand1); |
|  |  |
|  |  |
|  | /\*\* |
|  | \* finds square of operand1 and returns the result |
|  | \* @param[in] operand1 |
|  | \* @return floating point value of square of operand1 |
|  | \*/ |
|  | float square(float operand1); |
|  |  |
|  |  |
|  | /\*\* |
|  | \* finds cube of operand1 and returns the result |
|  | \* @param[in] operand1 |
|  | \* @return floating point value of cube of operand1 |
|  | \*/ |
|  | float cube(float operand1); |
|  | #endif /\* #define \_\_ADVANCED\_CALCULATOR\_H\_\_ \*/ |