|  |
| --- |
| implementation |
|  | #incude<unity.h>  #include <calculator.h> |
|  | #include <math.h> |
|  |  |
|  | static arthimetic\_inputs a\_i={0,0,0}; |
|  | static single\_inputs s\_i={0,0}; |
|  | static trignometric\_inputs t\_i={0,0}; |
|  |  |
|  | /\* Prototypes for all the test functions \*/ |
|  | void test\_add(void); |
|  | void test\_subtract(void); |
|  | void test\_multiply(void); |
|  | void test\_divide(void); |
|  | void test\_power(void); |
|  | void test\_squareroot(void); |
|  | void test\_logarithm(void); |
|  | void test\_factorial(void); |
|  | void test\_sine(void); |
|  | void test\_cosine(void); |
|  | void test\_tangent(void); |
|  | void test\_cotangent(void); |
|  | void test\_secant(void); |
|  | void test\_cosecant(void); |
|  |  |
|  | /\* Required by the unity test framework \*/ |
|  | void setUp(){} |
|  | /\* Required by the unity test framework \*/ |
|  | void tearDown(){} |
|  |  |
|  | /\* Start of the application test \*/ |
|  | int main() |
|  | { |
|  | /\* Initiate the Unity Test Framework \*/ |
|  | UNITY\_BEGIN(); |
|  |  |
|  | /\* Run Test functions \*/ |
|  | RUN\_TEST(test\_add); |
|  | RUN\_TEST(test\_subtract); |
|  | RUN\_TEST(test\_multiply); |
|  | RUN\_TEST(test\_divide); |
|  | RUN\_TEST(test\_power); |
|  | RUN\_TEST(test\_squareroot); |
|  | RUN\_TEST(test\_logarithm); |
|  | RUN\_TEST(test\_factorial); |
|  | RUN\_TEST(test\_sine); |
|  | RUN\_TEST(test\_cosine); |
|  | RUN\_TEST(test\_tangent); |
|  | RUN\_TEST(test\_cotangent); |
|  | RUN\_TEST(test\_secant); |
|  | RUN\_TEST(test\_cosecant); |
|  |  |
|  | /\* Close the Unity Test Framework \*/ |
|  | return UNITY\_END(); |
|  | } |
|  |  |
|  | /\* Write all the test functions \*/ |
|  | void test\_add(void) { |
|  | a\_i.input\_1=20; |
|  | a\_i.input\_2=10; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS, add(&a\_i)); |
|  | TEST\_ASSERT\_EQUAL(30,a\_i.output); |
|  |  |
|  | a\_i.input\_1=2\*3+5; |
|  | a\_i.input\_2=8/2-1; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS, add(&a\_i)); |
|  | TEST\_ASSERT\_EQUAL(14,a\_i.output); |
|  |  |
|  | a\_i.input\_1=9223372036854775807; |
|  | a\_i.input\_2=-2; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS, add(&a\_i)); |
|  | TEST\_ASSERT\_EQUAL((9223372036854775805),a\_i.output); |
|  | } |
|  |  |
|  | void test\_subtract(void) { |
|  | a\_i.input\_1=20; |
|  | a\_i.input\_2=10; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS, subtract(&a\_i)); |
|  | TEST\_ASSERT\_EQUAL(10,a\_i.output); |
|  |  |
|  | a\_i.input\_1=20/2+1; |
|  | a\_i.input\_2=10\*1; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS, subtract(&a\_i)); |
|  | TEST\_ASSERT\_EQUAL(1,a\_i.output); |
|  |  |
|  | a\_i.input\_1=9223372036854775807; |
|  | a\_i.input\_2=2; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS, subtract(&a\_i)); |
|  | TEST\_ASSERT\_EQUAL(9223372036854775805,a\_i.output); |
|  | } |
|  |  |
|  | void test\_multiply(void) { |
|  | a\_i.input\_1=20; |
|  | a\_i.input\_2=10; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS, multiply(&a\_i)); |
|  | TEST\_ASSERT\_EQUAL(200,a\_i.output); |
|  |  |
|  | a\_i.input\_1=2+6/2; |
|  | a\_i.input\_2=0; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS, multiply(&a\_i)); |
|  | TEST\_ASSERT\_EQUAL(0,a\_i.output); |
|  |  |
|  | a\_i.input\_1=999999; |
|  | a\_i.input\_2=123456; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS, multiply(&a\_i)); |
|  | TEST\_ASSERT\_EQUAL(123455876544,a\_i.output); |
|  | } |
|  |  |
|  | void test\_divide(void) { |
|  | a\_i.input\_1=20; |
|  | a\_i.input\_2=10; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS, division(&a\_i)); |
|  | TEST\_ASSERT\_EQUAL(2,a\_i.output); |
|  |  |
|  | /\* we are making output as '0' when input\_2 is '0' \*/ |
|  | a\_i.input\_1=20; |
|  | a\_i.input\_2=0; |
|  | TEST\_ASSERT\_EQUAL(ERROR\_DIVISION\_BY\_ZERO, division(&a\_i)); |
|  | TEST\_ASSERT\_EQUAL(0,a\_i.output); |
|  |  |
|  | a\_i.input\_1=9223372036854775806; |
|  | a\_i.input\_2=2; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS, division(&a\_i)); |
|  | TEST\_ASSERT\_EQUAL(4611686018427387903,a\_i.output); |
|  |  |
|  |  |
|  | } |
|  | void test\_power(void){ |
|  | a\_i.input\_1=2; |
|  | a\_i.input\_2=3; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS, power(&a\_i)); |
|  | TEST\_ASSERT\_EQUAL(8,a\_i.output); |
|  |  |
|  | a\_i.input\_1=2\*2+5\*23; |
|  | a\_i.input\_2=2; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS, power(&a\_i)); |
|  | TEST\_ASSERT\_EQUAL(14161,a\_i.output); |
|  | } |
|  | void test\_squareroot(void){ |
|  | s\_i.input\_3=9; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS,square\_root(&s\_i)); |
|  | TEST\_ASSERT\_EQUAL(3,s\_i.output\_2); |
|  |  |
|  | s\_i.input\_3=9223372036854775805; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS,square\_root(&s\_i)); |
|  | TEST\_ASSERT\_EQUAL(3037000448,s\_i.output\_2); |
|  | } |
|  | void test\_logarithm(void){ |
|  | s\_i.input\_3=10; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS,logarithm(&s\_i)); |
|  | TEST\_ASSERT\_EQUAL(1,s\_i.output\_2); |
|  |  |
|  | /\* we are making output as '0' when input is lessthan '0' \*/ |
|  | s\_i.input\_3=-10; |
|  | TEST\_ASSERT\_EQUAL(UNDEFINED,logarithm(&s\_i)); |
|  | TEST\_ASSERT\_EQUAL(0,s\_i.output\_2); |
|  |  |
|  | s\_i.input\_3=100000000000000000; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS,logarithm(&s\_i)); |
|  | TEST\_ASSERT\_EQUAL(17,s\_i.output\_2); |
|  | } |
|  | void test\_factorial(void){ |
|  | s\_i.input\_3=4; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS,factorial(&s\_i)); |
|  | TEST\_ASSERT\_EQUAL(24,s\_i.output\_2); |
|  |  |
|  | /\* we are making output as '0' when input is lessthan or equla to '0' \*/ |
|  | s\_i.input\_3=-1; |
|  | TEST\_ASSERT\_EQUAL(FAILURE,factorial(&s\_i)); |
|  | TEST\_ASSERT\_EQUAL(0,s\_i.output\_2); |
|  |  |
|  | } |
|  | void test\_sine(void){ |
|  | t\_i.input\_4=90; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS,sine(&t\_i)); |
|  | TEST\_ASSERT\_EQUAL(1,t\_i.output\_3); |
|  | } |
|  | void test\_cosine(void){ |
|  | t\_i.input\_4=0; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS,cosine(&t\_i)); |
|  | TEST\_ASSERT\_EQUAL(1,t\_i.output\_3); |
|  | } |
|  | void test\_tangent(void){ |
|  | t\_i.input\_4=45; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS,tangent(&t\_i)); |
|  | TEST\_ASSERT\_EQUAL(0.999999,t\_i.output\_3); |
|  |  |
|  | t\_i.input\_4=90; |
|  | TEST\_ASSERT\_EQUAL(UNDEFINED,tangent(&t\_i)); |
|  | TEST\_ASSERT\_EQUAL(2,t\_i.output\_3); |
|  | } |
|  | void test\_cotangent(void){ |
|  | t\_i.input\_4=45; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS,cotangent(&t\_i)); |
|  | TEST\_ASSERT\_EQUAL(1.000001,t\_i.output\_3); |
|  |  |
|  | t\_i.input\_4=90; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS,cotangent(&t\_i)); |
|  | TEST\_ASSERT\_EQUAL(0,t\_i.output\_3); |
|  | } |
|  | void test\_secant(void){ |
|  | t\_i.input\_4=0; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS,secant(&t\_i)); |
|  | TEST\_ASSERT\_EQUAL(1,t\_i.output\_3); |
|  |  |
|  | t\_i.input\_4=90; |
|  | TEST\_ASSERT\_EQUAL(UNDEFINED,secant(&t\_i)); |
|  | TEST\_ASSERT\_EQUAL(2,t\_i.output\_3); |
|  | } |
|  | void test\_cosecant(void){ |
|  | t\_i.input\_4=90; |
|  | TEST\_ASSERT\_EQUAL(SUCCESS,sine(&t\_i)); |
|  | TEST\_ASSERT\_EQUAL(1,t\_i.output\_3); |
|  |  |
|  | t\_i.input\_4=180; |
|  | TEST\_ASSERT\_EQUAL(UNDEFINED,cosecant(&t\_i)); |
|  | TEST\_ASSERT\_EQUAL(2,t\_i.output\_3); |
|  | } |