Problem 1a / 1b

The task is to implement a Simpsons Adaptive Algorithm in C++, and apply the algorithm to a test function. The function to integrate is the one suggested in the assignment:

$$\int_0^{\pi} [x + \cos(x^5)]\, dx$$

The algorithm is tested using a tolerance τ of 10e−2,10e−3,10e−4 and 10e-8. Results are compared with the values provided by MATLAB's integration functions using an absolute tolerance of 10e−8.

The count of function calls is implemented via a pointer to avoid using a global variable.

Command Line Output:

```
username@username-VirtualBox:~/Desktop/github/SF2565/Assign1/Problem1/Debug$ ./Assignment1
Result (10e-2) : 4.62617
Number of function calls : 1
Result (10e-3) : 5.98101
Number of function calls : 115
Result (10e-4) : 5.80506
Number of function calls : 361
Result (10e-8) : 5.80606
Number of function calls : 3871
```

	C++		Matlab
Tolerance	**End Result**	**No. Function Calls**	**End Result**
10e-2	4.62617	1	NA
10e-3	5.98101	115	NA
10e-4	5.80506	361	NA
10e-8	5.80606	3871	5.8061

The number of function calls grows exponentially with the reduction in tolerance.

Problem 2a / 2b

Unit testing is added to the application using utest.h. The following test cases are created and evaluated:

- Expected function value for a simple integrand:

  $\int_0^1 x\,dx = 1/2$
- Exception handling for a negative tolerance argument
- Exception handling for instance of a null pointer
- Exception handling for invalid limits of integration

All tests are successfully passed.

Command Line Output:

```
username@username-VirtualBox:~/Desktop/github/SF2565/Assign1/Problem2/Debug$ ./Assignment1_2
[==========] Running 4 test cases.
[ RUN      ] IS_HALF.TEST1
[       OK ] IS_HALF.TEST1 (892ns)
[ RUN      ] EXCEPTIONS.NEG_TOLERANCE
Exception caught: Negative Tolerance!
[       OK ] EXCEPTIONS.NEG_TOLERANCE (41535ns)
[ RUN      ] EXCEPTIONS.NULL_FUNC_POINTER
Exception caught: NULL Function Pointer!
[       OK ] EXCEPTIONS.NULL_FUNC_POINTER (4087ns)
[ RUN      ] EXCEPTIONS.INVALID_LIMITS
Exception caught: Invalid Integration Limits!
[       OK ] EXCEPTIONS.INVALID_LIMITS (2961ns)
[==========] 4 test cases ran.
[  PASSED  ] 4 tests.
```

Appendix

Attached below are code file descriptions:

1. p1_main.cpp - main code for Problem 1
2. P2_main.cpp - main code for Problem 2
3. adaptive_integration.cpp/.hpp - Actual algorithms code
4. P1_integrationscript.m - matlab code to compare results in Problem 1

```cpp
/*
 * main.cpp
 *
 *  Created on: Sep 9, 2024
 *      Author: Shawn/Alessio
 */


#include <iostream>
#include "adaptive_integration.hpp"

using namespace std;

// Function to test
double func1 (double x){
        return std::abs(x + cos(pow(x,5)));
}

int main(){

        uint32_t func_counter = 0;
        double result = 0;
        // Call recursive function
        result = func_ASI(func1, 0, M_PI, 10e-2, func_counter);
        cout << "Result (10e-2) : " << result << endl;
        cout << "Number of function calls : " << func_counter << endl;

        func_counter = 0;
        result = 0;
        // Call recursive function
        result = func_ASI(func1, 0, M_PI, 10e-3, func_counter);
        cout << "Result (10e-3) : " << result << endl;
        cout << "Number of function calls : " << func_counter << endl;

        func_counter = 0;
        result = 0;
        // Call recursive function
        result = func_ASI(func1, 0, M_PI, 10e-4, func_counter);
        cout << "Result (10e-4) : " << result << endl;
        cout << "Number of function calls : " << func_counter << endl;

        func_counter = 0;
        result = 0;
        // Call recursive function
        result = func_ASI(func1, 0, M_PI, 10e-8, func_counter);
        cout << "Result (10e-8) : " << result << endl;
        cout << "Number of function calls : " << func_counter << endl;

        return 1;
}
```

```cpp
/*
 * main.cpp
 *
 *  Created on: Sep 16, 2024
 *      Author: root
 */

#include <cmath>
#include <vector>

#include "utest.h"
#include "adaptive_integration.hpp"             //Include functions to be tested

using namespace std;

UTEST_STATE();          //instantiate test cases and build test framework

uint32_t func_count = 0;        // Counter for number of function calls // UNUSED

// Function to test
double func_x(double x){
        return x;
}

UTEST(IS_HALF, TEST1){
        ASSERT_EQ(0.5, func_ASI(func_x, 0, 1, 0.005, func_count));
}

UTEST(EXCEPTIONS, NEG_TOLERANCE){
        ASSERT_EQ(-1, func_ASI(func_x, 0, 1, -0.005, func_count));
}

UTEST(EXCEPTIONS, NULL_FUNC_POINTER){
        ASSERT_EQ(-1, func_ASI(0, 0, 1, 0.005, func_count));
}

UTEST(EXCEPTIONS, INVALID_LIMITS){
        ASSERT_EQ(-1, func_ASI(func_x, 1, 0, 0.005, func_count));
}


// Run test cases
int main(int argc, const char *const argv[]){

        return utest_main(argc, argv);  // Call test framework
}
```

```cpp
/*
 * adaptive_integration.cpp
 *
 *  Created on: Sep 16, 2024
 *      Author: Shawn / Alessio
 */

#include "adaptive_integration.hpp"
#include <stdexcept>
#include <iostream>

using namespace std;

/*Function Description: Calls Simpsons Integration function
 *
 *Parameters:   func_f - Math function to integrate
 *                       begin_limit - Initial Limit
 *                       end_limit - Final limit
 *
 *Returns:              Integration value
 * */
double func_simpsons_rule( const function<double(double)>& func_f,
                double begin_limit,
                double end_limit) {

        double midpoint = (begin_limit + end_limit) / 2;
        return (end_limit - begin_limit) / 6 * (func_f(begin_limit) + 4 * func_f(midpoint)
+ func_f(end_limit));

}

/*Function Description: Adaptive Simpsons Integration.
 *                                          Calls Simpsons Integration function
recursively.
 *
 *Parameters:   func_f - Math function to integrate
 *                       begin_limit - Initial Limit
 *                       end_limit - Final limit
 *                       tolerance - Minimum tolerance required for final result
 *                       func_call_counter - Number of function calls made
 *
 *Returns:              Integration value - on Success
 *                       -1 - On Failure
 * */
double func_ASI(const function<double(double)>& func_f,
                double begin_limit,
                double end_limit,
                double tolerance,
                uint32_t& func_call_counter) {

        // Initialisations
        double I1 = 0;  // I1 integration I(α,β)
        double I2 = 0;  // I2 integration  I2(α,β)
        double midpoint = 0;    // midpoint calculation for I2 (γ)
        double errest = 0;              // error of simpsons calculation

        // Input validation
        try{
                if (tolerance <= 0){            // Check tolerance is positive
                        throw std::runtime_error("Negative Tolerance!");
                }

                if (!func_f) {                  // Check for null pointer
                        // TODO need an error handler for: ptr == nullptr. See slides
lecture 3 page 13
                        throw std::runtime_error("NULL Function Pointer!");
```

```
                }

                if (begin_limit > end_limit) {          // Check if begin_limit <
end_limit, else return with -1
                        throw std::runtime_error("Invalid Integration Limits!");
                }

        }

        catch (const std::runtime_error& e) {
                // Handle the exception
                std::cerr << "Exception caught: " << e.what() << std::endl;
                return -1;
        }

        // Increment function counter
        func_call_counter++;

        // Calculate I1 (Call func_simpsons_rule)
        I1 = func_simpsons_rule(func_f, begin_limit, end_limit);

        midpoint = (begin_limit + end_limit) / 2;               // Calculate half
intervals  γ = 1 / 2 * (α+β)

        // Calculate I2 [ I2(α,β) := I(α,γ) + I(γ,β)]
        I2 = func_simpsons_rule(func_f, begin_limit, midpoint)
                + func_simpsons_rule(func_f, midpoint, end_limit);

        // Error estimate
        errest = std::abs(I2 - I1);

        // Check if error estimate is within tolerance
        if (errest < 15 * tolerance) {
                return I2;
        }

        // Call function again [ASI(f,a,γ,τ/2) + ASI(f,γ,b,τ/2)]
        return func_ASI(func_f, begin_limit, midpoint, tolerance / 2, func_call_counter)
                        + func_ASI(func_f, midpoint, end_limit, tolerance /
2,func_call_counter);
}
```

```
/*
 * adaptive_integration.h
 *
 *  Created on: Sep 16, 2024
 *      Author: Shawn / Alessio
 */

#ifndef ADAPTIVE_INTEGRATION_HPP_
#define ADAPTIVE_INTEGRATION_HPP_

#include <cmath>
#include <stdint.h>
#include <vector>
#include <functional>

using namespace std;

double func_simpsons_rule(const function<double(double)>& func_f, double begin_limit,
double end_limit, double tolerance);
double func_ASI(const function<double(double)>& func_f, double begin_limit, double
end_limit, double tolerance, uint32_t& func_call_counter);

#endif /* ADAPTIVE_INTEGRATION_HPP_ */
```

```
% Define the integrand as an anonymous function
f = @(x) x + cos(x.^5);

% Set the limits of integration
a = 0;
b = pi;

% Perform the integration using MATLAB's integral function
result = integral(f, a, b);

% Display the result
disp(result);
```