```
/*
 * 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);
}
```