Практическая работа № 3.  
Многопоточность  
В математических расчетах

**Цель практической работы**

Закрепление теоретических знаний по основам разработки многопоточных приложений.

**Постановка задачи**

**Поток** — это основная единица, которой операционная система выделяет время процессора. При помощи многопоточности можно выделить в приложении несколько потоков, которые будут выполнять различные задачи одновременно.

В многопоточном режиме в рамках одного процесса решаются несколько задач разными потоками.

**Основные свойства класса Thread:**

статическое свойство **CurrentThread** позволяет получить текущий поток;

**IsAlive**: указывает, работает ли поток в текущий момент;

**Name**: содержит имя потока;

**ManagedThreadId**: возвращает числовой идентификатор текущего потока;

**Priority**: хранит приоритет потока - значение перечисления **ThreadPriority**:

* **Lowest**;
* **BelowNormal**;
* **Normal** (по умолчанию);
* **AboveNormal**;
* **Highest**.

Получение информации о текущем потоке (рисунок 1):

using System.Threading;

// получаем текущий поток

Thread currentThread = Thread.CurrentThread;

//получаем имя потока

Console.WriteLine($"Имя потока: {currentThread.Name}");

currentThread.Name = "Метод Main";

Console.WriteLine($"Имя потока: {currentThread.Name}");

Console.WriteLine($"Запущен ли поток: {currentThread.IsAlive}");

Console.WriteLine($"Id потока: {currentThread.ManagedThreadId}");

Console.WriteLine($"Приоритет потока: {currentThread.Priority}");

Console.WriteLine($"Статус потока: {currentThread.ThreadState}");

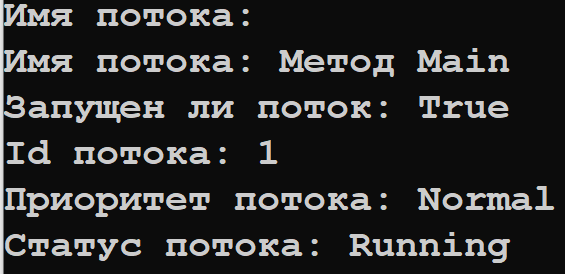


Рисунок 1 – Отображение информации о потоке

Класс **Thread** определяет ряд методов для управления потоком. Основные из них:

* Статический метод **Sleep** останавливает поток на определенное количество миллисекунд;
* Метод **Interrupt** прерывает поток, который находится в состоянии WaitSleepJoin;
* Метод **Join** блокирует выполнение вызвавшего его потока до тех пор, пока не завершится поток, для которого был вызван данный метод;
* Метод **Start** запускает поток.

**Примеры** определения потоков (рисунок 2):

Thread myThread1 = new Thread(Print);

Thread myThread2 = new Thread(new ThreadStart(Print));

Thread myThread3 = new Thread(() => Console.WriteLine("Hello Threads"));

void Print()

{

Thread.Sleep(2000);

Console.WriteLine($"Hello Threads

{Thread.GetCurrentProcessorId()}");

}

Для запуска нового потока применяется метод **Start** класса **Thread**:

myThread1.Start(); // запускаем поток myThread1

myThread2.Start(); // запускаем поток myThread2

myThread3.Start(); // запускаем поток myThread3

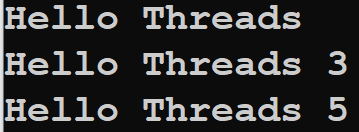


Рисунок 2 – Запуск 3 потоков

**Пример** запуска потоков в цикле (рисунок 3):

using System.Threading;

// создаем новый поток

Thread myThread = new Thread(Print);

myThread.Start(); // запускаем поток myThread

for (int i = 0; i < 5; i++)

{// действия, выполняемые в главном потоке

Console.WriteLine($"Главный поток: {i}");

Thread.Sleep(400);

}

void Print()

{// действия, выполняемые во втором потоке

for (int i = 0; i < 5; i++)

{

Console.WriteLine($"Второй поток: {i}");

Thread.Sleep(400);

}

}

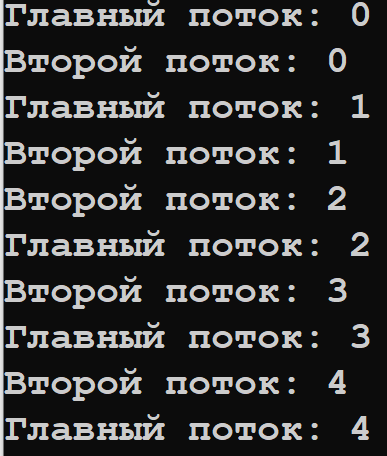


Рисунок 3 – Запуск потоков в цикле

Когда два или более потоков имеют доступ к одному разделенному ресурсу, они нуждаются в обеспечении того, что ресурс будет использован только одним потоком одновременно. Процесс, с помощью которого это достигается, называется **синхронизацией**.

Чтобы синхронизировать потоки и ограничить доступ к разделяемым ресурсам на время их использования каким-нибудь потоком. Для этого используется ключевое слово **lock**. Оператор **lock** определяет блок кода, внутри которого весь код блокируется и становится недоступным для других потоков до завершения работы текущего потока. Остальные потоки помещаются в очередь ожидания и ждут, пока текущий поток не освободит данный блок кода.

**Пример** многопоточного расчета числа пи (рисунок 4):

var iterations = (long)1e8;

// объект-заглушка

object locker = new();

var pi = 0.0;

CalculatePiMultithread(iterations);

Console.WriteLine(pi);

void CalculatePiThread(long from, long to)

{

for (long i = from; i < to; i++)

{

var sign = i % 2 == 0 ? 1 : -1;

var s = 4 \* sign / (2 \* i + 1.0);

lock (locker)

pi += s;

}

Console.WriteLine($"{Thread.CurrentThread.Name} Готово!");

}

double CalculatePiMultithread(long n)

{

var threadsCount = Environment.ProcessorCount;

var threads = new Thread[threadsCount];

var iterationsOnThread = (double)n / threadsCount;

for (int i = 0; i < threadsCount; i++)

{

var from = (long)(i \* iterationsOnThread);

var to = (long)((i + 1) \* iterationsOnThread);

Thread myThread = new(() => CalculatePiThread(from, to));

myThread.Name = $"Поток {i}";

threads[i] = myThread;

myThread.Start();

}

//Ждем пока потоки досчитают

foreach (var thread in threads)

thread.Join();

return pi;

}

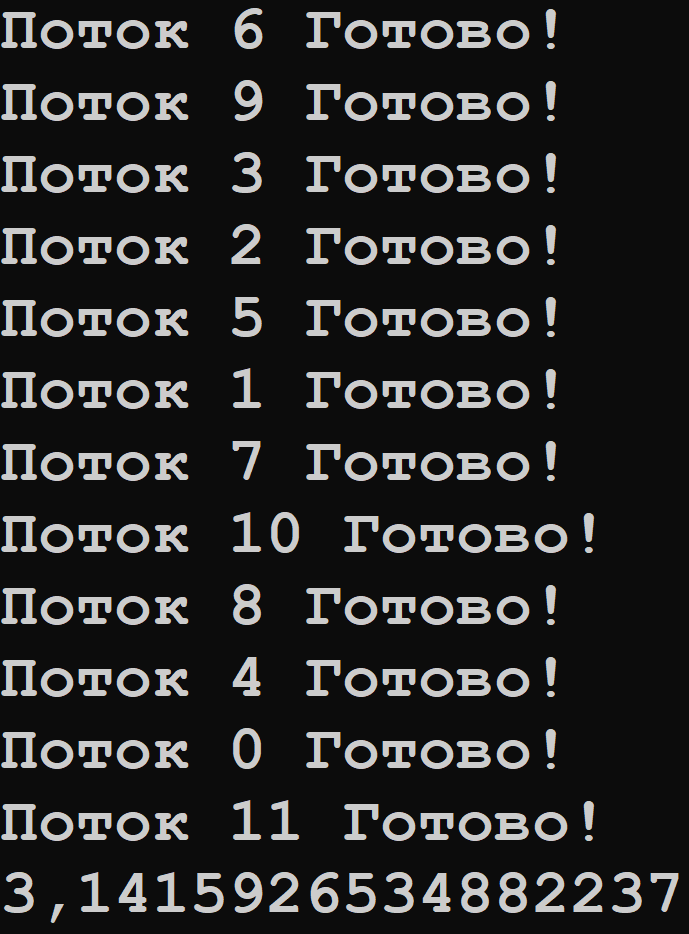


Рисунок 4 – Результат многопоточного расчета числа пи

**Задание на практическую работу**

1. В соответствии с вариантом разработать многопоточное графическое приложение для расчета интеграла.
2. Вычисление интеграла должно проводиться в отдельном потоке, чтобы программа не «зависала» на время расчета.
3. Предусмотреть динамическое отображение прогресса с помощью progressBar.
4. Предусмотреть задание произвольных пределов интегрирования и числа разбиений области интегрирования.
5. Защита работы включает демонстрацию работы программы, сравнение полученного значения интеграла с аналитическим решением.

При вычислении интеграла разбить исходный интеграл на сумму интегралов по более маленьким областям:

,

где , , *N* – количество разбиений.

Для вычисления интеграла использовать [квадратурный метод Гаусса-3](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D1%83%D1%80%D0%BD%D1%8B%D1%85_%D1%84%D0%BE%D1%80%D0%BC%D1%83%D0%BB):

,

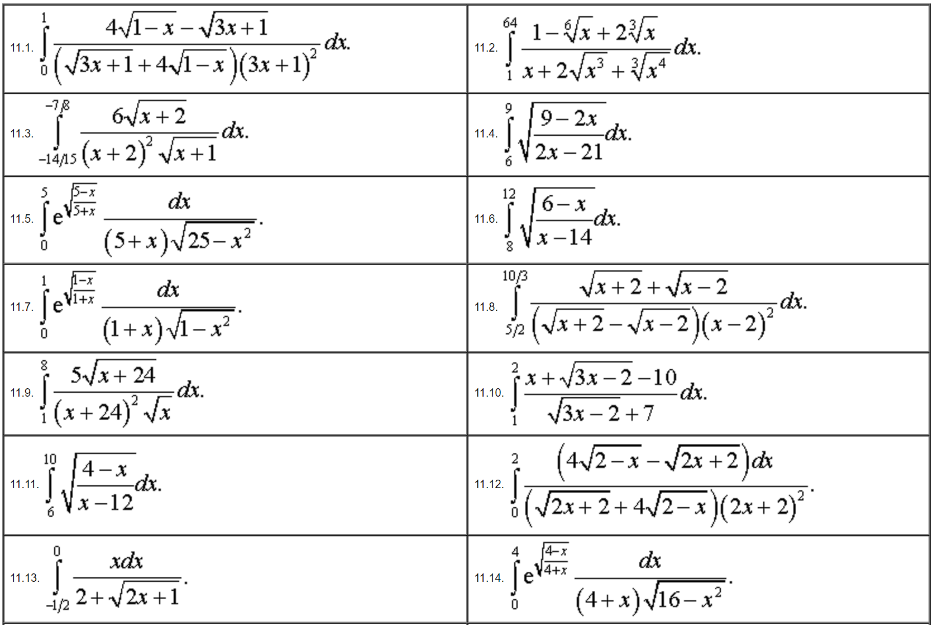
где

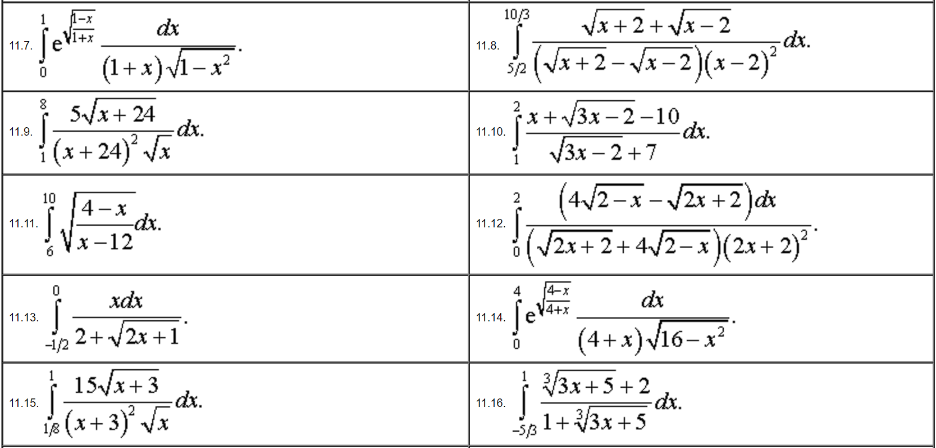
,

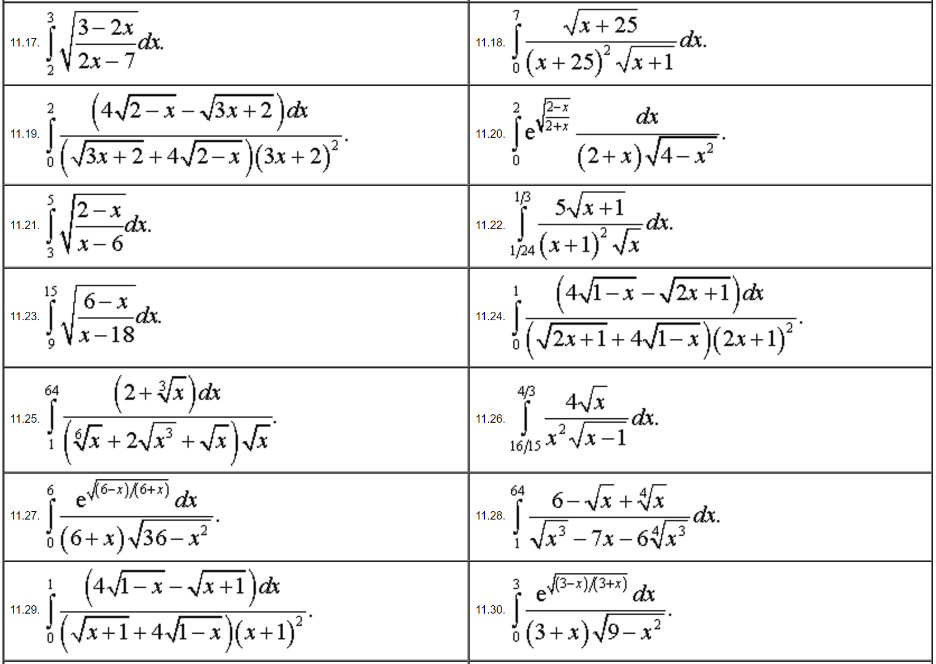
, , ,

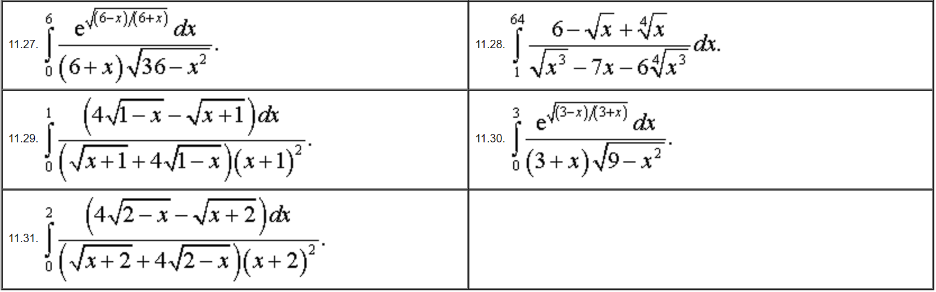
, , .

**Варианты заданий на практическую работу**









**Пример разработки приложения**

Напишем программу для интегрирования методом Гаусса-2. Функция y = x2.

**Шаг 1: Создание проекта Windows Forms (.NET 8)**

1. Откройте Visual Studio
2. Создайте новый проект:
   * Выберите "Windows Forms App" (.NET 8.0)
   * Назовите проект "IntegralCalculatorApp"
3. Убедитесь, что выбран .NET 8.0 в выпадающем списке

**Шаг 2: Создание класса для математических расчетов**

GaussIntegralCalculator.cs:

namespace IntegralCalculatorApp;

public record IterationSolvedEvemtArgs(int Iteration, double Sum);

public delegate void IterationSolved(object sender, IterationSolvedEvemtArgs args);

public class GaussIntegralCalculator

{

// Точки и веса для метода Гаусса-2

private static readonly double[] GaussPoints = { -1.0 / Math.Sqrt(3), 1.0 / Math.Sqrt(3) };

private static readonly double[] GaussWeights = { 1.0, 1.0 };

public event IterationSolved? IterationSolved;

public double CalculateIntegral(Func<double, double> f, double a, double b, int segments)

{

if (segments <= 0)

throw new ArgumentException("Количество сегментов должно быть положительным");

double segmentWidth = (b - a) / segments;

double sum = 0.0;

for (int i = 0; i < segments; i++)

{

double segmentA = a + i \* segmentWidth;

double segmentB = segmentA + segmentWidth;

sum += CalculateSegmentIntegral(f, segmentA, segmentB);

IterationSolved?.Invoke(this, new IterationSolvedEvemtArgs(i, sum));

}

return sum;

}

private double CalculateSegmentIntegral(Func<double, double> f, double a, double b)

{

double sum = 0.0;

double detJ = (b - a) / 2;

double halfLength = detJ;

for (int i = 0; i < GaussPoints.Length; i++)

{

double x = a + halfLength \* (GaussPoints[i] + 1);

sum += detJ \* GaussWeights[i] \* f(x);

}

return sum;

}

}

**Шаг 3: Дизайн главной формы**

1. Откройте Form1.cs в конструкторе форм
2. Добавьте следующие элементы:
   * Label "Нижний предел:"
   * NumericUpDown для нижнего предела (numLowerLimit)
   * Label "Верхний предел:"
   * NumericUpDown для верхнего предела (numUpperLimit)
   * Label "Количество сегментов:"
   * NumericUpDown для количества сегментов (numSegments)
   * Button "Вычислить" (btnCalculate)
   * ProgressBar для отображения прогресса (progressBar)
   * Label для вывода результата (lblResult)

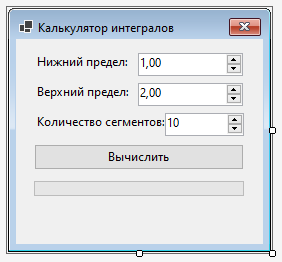


Рисунок 5 – Главное окно программы

**Шаг 4: Код главной формы (Form1.cs)**

using System.Threading;

namespace IntegralCalculatorApp

{

public partial class Form1 : Form

{

private Thread? \_calculationThread;

private bool \_calculationRunning;

private readonly object \_lockObject = new();

public Form1()

{

InitializeComponent();

}

private void btnCalculate\_Click(object sender, EventArgs e)

{

if (\_calculationRunning)

{

MessageBox.Show("Вычисление уже выполняется", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

return;

}

double a = (double)numLowerLimit.Value;

double b = (double)numUpperLimit.Value;

int segments = (int)numSegments.Value;

if (b <= a)

{

MessageBox.Show("Верхний предел должен быть больше нижнего", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

\_calculationRunning = true;

btnCalculate.Enabled = false;

progressBar.Value = 0;

lblResult.Text = "Вычисление...";

\_calculationThread = new Thread(() => CalculateIntegral(a, b, segments));

\_calculationThread.Start();

}

private void CalculateIntegral(double a, double b, int segments)

{

var gic = new GaussIntegralCalculator();

gic.IterationSolved += (s, a) =>

{

int progress = (a.Iteration + 1) \* 100 / segments;

UpdateProgress(progress, a.Sum);

};

var result = gic.CalculateIntegral(x => x \* x, a, b, segments);

if (\_calculationRunning)

{

UpdateResult(result);

\_calculationRunning = false;

}

}

private void UpdateProgress(int progress, double currentValue)

{

if (InvokeRequired)

{

Invoke(new Action(() => UpdateProgress(progress, currentValue)));

return;

}

progressBar.Value = progress;

lblResult.Text = $"Текущее значение: {currentValue:F16}";

}

private void UpdateResult(double result)

{

if (InvokeRequired)

{

Invoke(new Action(() => UpdateResult(result)));

return;

}

lblResult.Text = $"Результат: {result:F16}";

btnCalculate.Enabled = true;

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

\_calculationRunning = false;

\_calculationThread?.Join();

}

}

}

**Шаг 5: Код дизайнера формы (Form1.Designer.cs)**

namespace IntegralCalculatorApp

{

partial class Form1

{

private System.ComponentModel.IContainer components = null;

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

private void InitializeComponent()

{

label1 = new Label();

numLowerLimit = new NumericUpDown();

label2 = new Label();

numUpperLimit = new NumericUpDown();

label3 = new Label();

numSegments = new NumericUpDown();

btnCalculate = new Button();

progressBar = new ProgressBar();

lblResult = new Label();

((System.ComponentModel.ISupportInitialize)numLowerLimit).BeginInit();

((System.ComponentModel.ISupportInitialize)numUpperLimit).BeginInit();

((System.ComponentModel.ISupportInitialize)numSegments).BeginInit();

SuspendLayout();

//

// label1

//

label1.AutoSize = true;

label1.Location = new Point(18, 15);

label1.Name = "label1";

label1.Size = new Size(98, 15);

label1.TabIndex = 0;

label1.Text = "Нижний предел:";

//

// numLowerLimit

//

numLowerLimit.DecimalPlaces = 2;

numLowerLimit.Increment = new decimal(new int[] { 1, 0, 0, 131072 });

numLowerLimit.Location = new Point(122, 14);

numLowerLimit.Margin = new Padding(3, 2, 3, 2);

numLowerLimit.Maximum = new decimal(new int[] { 10000, 0, 0, 0 });

numLowerLimit.Minimum = new decimal(new int[] { 10000, 0, 0, int.MinValue });

numLowerLimit.Name = "numLowerLimit";

numLowerLimit.Size = new Size(105, 23);

numLowerLimit.TabIndex = 1;

numLowerLimit.Value = new decimal(new int[] { 1, 0, 0, 0 });

//

// label2

//

label2.AutoSize = true;

label2.Location = new Point(18, 45);

label2.Name = "label2";

label2.Size = new Size(99, 15);

label2.TabIndex = 2;

label2.Text = "Верхний предел:";

//

// numUpperLimit

//

numUpperLimit.DecimalPlaces = 2;

numUpperLimit.Increment = new decimal(new int[] { 1, 0, 0, 131072 });

numUpperLimit.Location = new Point(122, 44);

numUpperLimit.Margin = new Padding(3, 2, 3, 2);

numUpperLimit.Maximum = new decimal(new int[] { 10000, 0, 0, 0 });

numUpperLimit.Minimum = new decimal(new int[] { 10000, 0, 0, int.MinValue });

numUpperLimit.Name = "numUpperLimit";

numUpperLimit.Size = new Size(105, 23);

numUpperLimit.TabIndex = 3;

numUpperLimit.Value = new decimal(new int[] { 2, 0, 0, 0 });

//

// label3

//

label3.AutoSize = true;

label3.Location = new Point(18, 75);

label3.Name = "label3";

label3.Size = new Size(135, 15);

label3.TabIndex = 4;

label3.Text = "Количество сегментов:";

//

// numSegments

//

numSegments.Location = new Point(149, 74);

numSegments.Margin = new Padding(3, 2, 3, 2);

numSegments.Maximum = new decimal(new int[] { 100000000, 0, 0, 0 });

numSegments.Minimum = new decimal(new int[] { 1, 0, 0, 0 });

numSegments.Name = "numSegments";

numSegments.Size = new Size(79, 23);

numSegments.TabIndex = 5;

numSegments.Value = new decimal(new int[] { 10, 0, 0, 0 });

//

// btnCalculate

//

btnCalculate.Location = new Point(18, 105);

btnCalculate.Margin = new Padding(3, 2, 3, 2);

btnCalculate.Name = "btnCalculate";

btnCalculate.Size = new Size(210, 26);

btnCalculate.TabIndex = 6;

btnCalculate.Text = "Вычислить";

btnCalculate.UseVisualStyleBackColor = true;

btnCalculate.Click += btnCalculate\_Click;

//

// progressBar

//

progressBar.Location = new Point(18, 142);

progressBar.Margin = new Padding(3, 2, 3, 2);

progressBar.Name = "progressBar";

progressBar.Size = new Size(210, 15);

progressBar.TabIndex = 7;

//

// lblResult

//

lblResult.AutoSize = true;

lblResult.Location = new Point(18, 172);

lblResult.Name = "lblResult";

lblResult.Size = new Size(0, 15);

lblResult.TabIndex = 8;

//

// Form1

//

AutoScaleDimensions = new SizeF(7F, 15F);

AutoScaleMode = AutoScaleMode.Font;

ClientSize = new Size(247, 205);

Controls.Add(lblResult);

Controls.Add(progressBar);

Controls.Add(btnCalculate);

Controls.Add(numSegments);

Controls.Add(label3);

Controls.Add(numUpperLimit);

Controls.Add(label2);

Controls.Add(numLowerLimit);

Controls.Add(label1);

FormBorderStyle = FormBorderStyle.FixedSingle;

Margin = new Padding(3, 2, 3, 2);

MaximizeBox = false;

MinimizeBox = false;

Name = "Form1";

StartPosition = FormStartPosition.CenterScreen;

Text = "Калькулятор интегралов";

FormClosing += Form1\_FormClosing;

((System.ComponentModel.ISupportInitialize)numLowerLimit).EndInit();

((System.ComponentModel.ISupportInitialize)numUpperLimit).EndInit();

((System.ComponentModel.ISupportInitialize)numSegments).EndInit();

ResumeLayout(false);

PerformLayout();

}

#endregion

private Label label1;

private NumericUpDown numLowerLimit;

private Label label2;

private NumericUpDown numUpperLimit;

private Label label3;

private NumericUpDown numSegments;

private Button btnCalculate;

private ProgressBar progressBar;

private Label lblResult;

}

}

**Шаг 6: Объяснение ключевых моментов**

1. **Многопоточность с Thread**:
   * Используем класс Thread для выполнения вычислений в фоновом потоке
   * Контролируем выполнение с помощью флага \_calculationRunning
   * Обеспечиваем безопасное завершение при закрытии формы
2. **Обновление UI из фонового потока**:
   * Используем InvokeRequired и Invoke для безопасного обновления элементов управления
   * Методы UpdateProgress и UpdateResult обеспечивают потокобезопасность
3. **Обработка ошибок**:
   * Проверка корректности введенных данных
   * Защита от повторного запуска вычислений
4. **Прогресс выполнения**:
   * Отображаем прогресс вычислений через ProgressBar
   * Показываем промежуточные результаты

**Шаг 7: Тестирование приложения**

1. Запустите приложение
2. Введите:
   * Нижний предел: 1
   * Верхний предел: 2
   * Количество сегментов: 100
3. Нажмите "Вычислить"
4. Убедитесь, что:
   * Прогресс-бар заполняется плавно
   * Результат ≈ 2.333 (точное значение 7/3)
   * Интерфейс остается отзывчивым
   * Можно безопасно закрыть форму во время вычислений

# Содержание пояснительной записки

1. Постановка задачи. Приводится теоретический материал, использованный при написании приложения.

2. Формулировка задания и вариант. Приводится задание на лабораторную работу и вариант этого задания.

3. Описание выполняемых действий. Необходимо привести описание последовательности разработки программы, реализации используемых методов, алгоритмов, блок-схем.

4. Анализ результатов. Привести анализ входных и выходных данных. Показать результаты выполнения программного кода. Предоставить скриншоты обработки тестовых примеров. Сделать выводы.

5. Листинг программы. Привести листинг разработанного программного кода, содержание файлов входных и выходных данных.

# Используемое программное обеспечение

1. Среда программирования MS Visual Studio Community 2022 (Свободно распространяемое программное обеспечение (в учебных целях));
2. Microsoft Office Standard 2007 (Open License: 42267924);
3. Open Office (Свободно распространяемое программное обеспечение).
4. Браузер (Свободно распространяемое программное обеспечение).

# Список литературы

* + - 1. Мейер Б. Объектно-ориентированное программирование и программная инженерия [Электронный ресурс]/ Мейер Б. – Электрон. текстовые данные. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. – 285 c.
      2. Биллиг, В. A. Основы объектного программирования на С# (C# 3.0, Visual Studio 2008) [Электронный ресурс] : учебное пособие / В. A. Биллиг. — Электрон. текстовые данные. — Москва, Саратов : Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2017. — 583 c. — 978-5-4487-0145-0. — Режим доступа: http://www.iprbookshop.ru/72339.html
      3. Павловская, Т. А. Программирование на языке высокого уровня C# [Электронный ресурс] / Т. А. Павловская. — 2-е изд. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 245 c. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/73713.html
      4. Агапов, В. П. Основы программирования на языке С# [Электронный ресурс] : учебное пособие / В. П. Агапов. — Электрон. текстовые данные. — М. : Московский государственный строительный университет, ЭБС АСВ, 2012. — 128 c. — 978-5-7264-0576-6. — Режим доступа: http://www.iprbookshop.ru/16366.html
      5. Медведев, М. А. Программирование на СИ# [Электронный ресурс] : учебное пособие / М. А. Медведев, А. Н. Медведев ; под ред. А. В. Присяжный. — Электрон. текстовые данные. — Екатеринбург : Уральский федеральный университет, ЭБС АСВ, 2015. — 64 c. — 978-5-7996-1561-1. — Режим доступа: http://www.iprbookshop.ru/69667.html
      6. Казанский А.А. Объектно-ориентированное программирование на языке Microsoft Visual С# в среде разработки Microsoft Visual Studio 2008 и .NET Framework. 4.3 [Электронный ресурс]: учебное пособие и практикум/ Казанский А.А.— Электрон. текстовые данные.— М.: Московский государственный строительный университет, ЭБС АСВ, 2011.— 180 c
      7. Уйманова Н.А. Основы объектно-ориентированного программирования [Электронный ресурс]: практикум/ Уйманова Н.А., Таспаева М.Г.— Электрон. текстовые данные.— Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2017.— 156 c.— Режим доступа: http://www.iprbookshop.ru/78808.html.— ЭБС «IPRbooks»
      8. Новиков П.В. Объектно-ориентированное программирование [Электронный ресурс]: учебно-методическое пособие к лабораторным работам/ Новиков П.В.— Электрон. текстовые данные.— Саратов: Вузовское образование, 2017.— 124 c.— Режим доступа: http://www.iprbookshop.ru/64650.html.— ЭБС «IPRbooks»