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}");

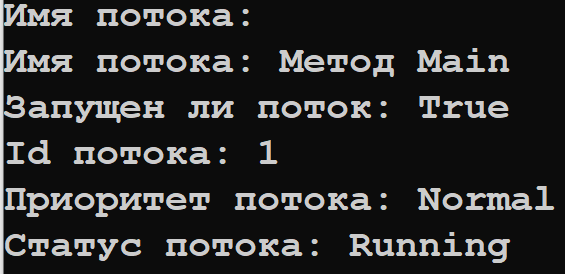


Рисунок 1 – Отображение информации о потоке

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

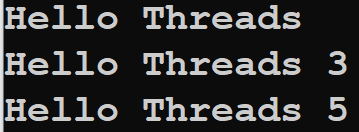


Рисунок 2 – Запуск 3 потоков

**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);

}

}

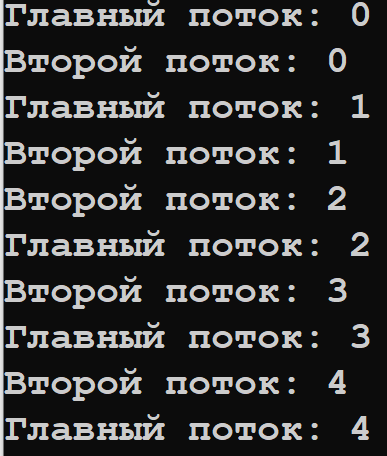


Рисунок 3 – Запуск потоков в цикле

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;

}

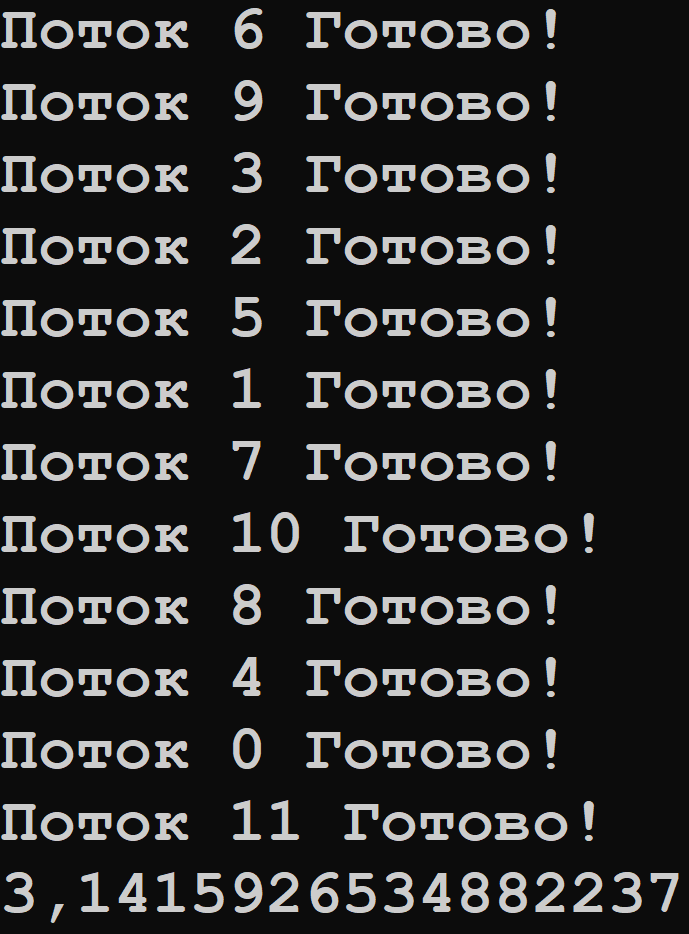


Рисунок 4 – Результат многопоточного расчета числа пи

**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):

,

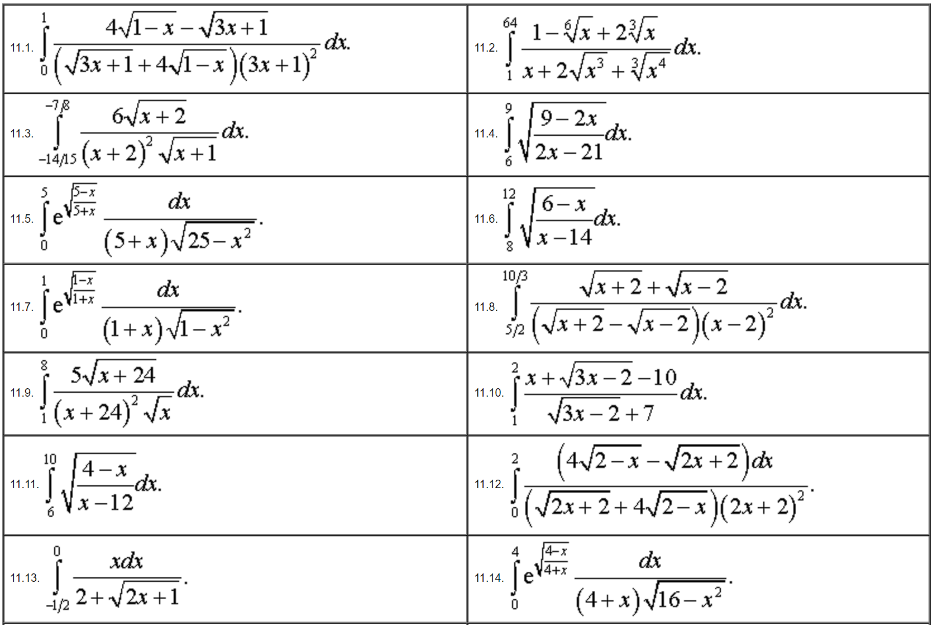
где

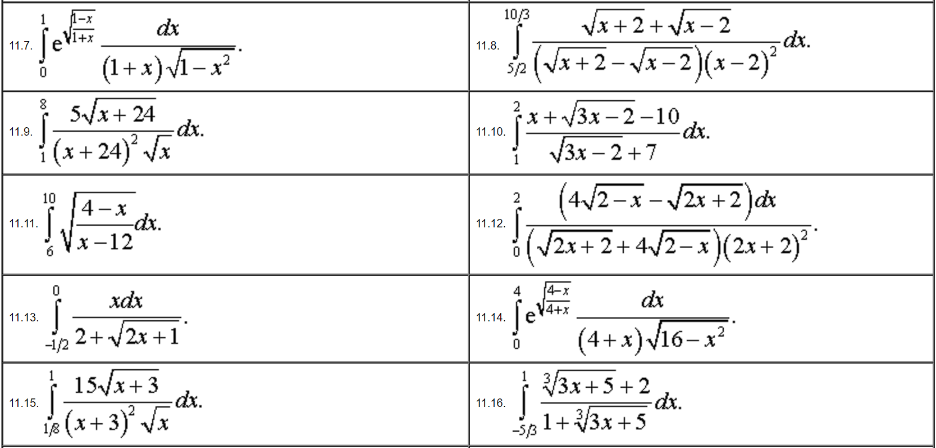
,

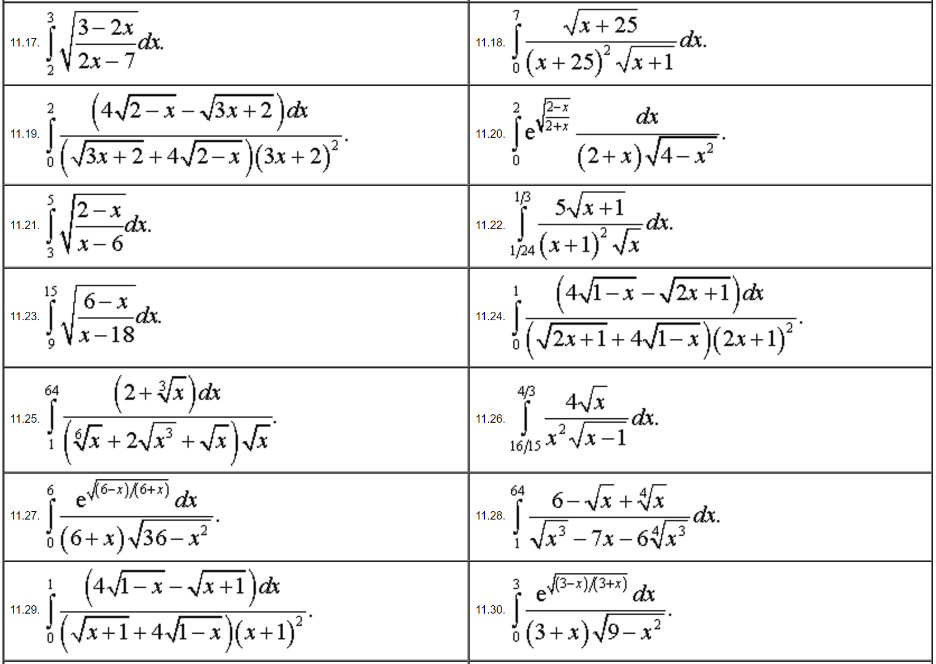
, , ,

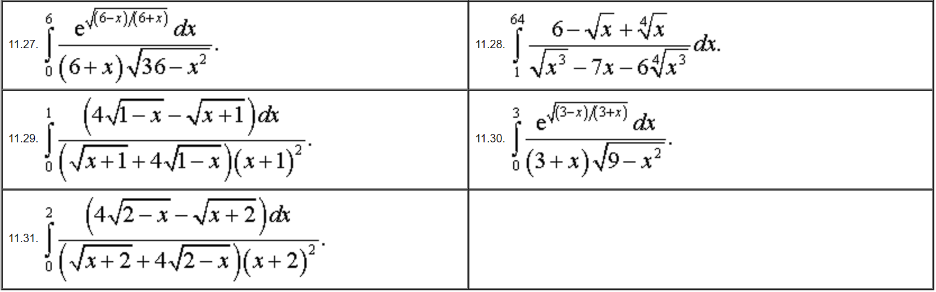
, , .

**Варианты заданий на практическую работу**









# 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»