Лабораторная работа 1. Вариант 2

Механизм привязки данных в Windows Presentation Foundation

В лабораторной работе надо создать пользовательский интерфейс приложения для накопления в коллекции результатов сравнения времени и точности вычислений значений математических функций **vmdTan и vmdErfInv** из пакета векторной математики библиотеки **Intel MKL** для различных режимов точности. Пользовательский интерфейс приложения дает возможность добавлять результаты в коллекцию, сохранять коллекцию в файле, загружать коллекцию из файла.

Функция **vmdErfInv** из пакета векторной математики библиотеки **Intel MKL** вычисляет значения, обратные к значениям функции ошибки. В стандартной библиотеке C++ есть функция **double erf (double)**, которая вычисляет функцию ошибки.

В среде VisualStudio надо создать решение (solution) с тремя проектами:

- тип одного проекта – **библиотека классов (class library) C#**, в которой находятся типы **WMf, VMGrid, VMTime, VMAccuracy, VMBenchmark;**
- тип второго проекта – **Dll-библиотека C++;**
- тип третьего проекта – приложение **Windows Presentation Foundation (WPF).**

Dll−библиотека C++

В Dll−библиотеке **C++** находится глобальная функция, которая вызывается из кода **C#** с использованием механизма **PInvoke.** Из этой глобальной функции вызываются функции из библиотеки **Intel MKL**.

Из кода **C#** через параметры глобальной функции передаются данные, которые необходимы для сравнения времени и точности вычислений. Через параметры глобальной функции в код **C#** возвращается код ошибки и результаты, необходимые для сравнения.

Библиотека классов C#

Библиотека классов C# содержит следующие типы.

Перечисление (**enum**) **VMf** с элементами, отвечающим функциям из пакета векторной математики **Intel MKL**, которые можно вызвать при работе приложения. Элементы перечисления используются как значения параметров некоторых методов.

Класс class VMGrid для параметров сетки, узлы которой являются элементами вектора аргументов тестируемой функции. Используется равномерная сетка, задаются концы отрезка (первый и последний узел) и число узлов. Для **VMGrid** используется тип **class,** так как тип **class** проще использовать в привязке к свойствам элементов управления.

В классе **VMGrid** определены открытые свойства

- типа **int** для длины вектора аргументов функции;
- два свойства типа **double** (или одно свойство типа **double[]**) для концов отрезка;
- свойство типа **double** для шага сетки (только с методом get).

В класс **VMGrid** можно добавить свойство типа **VMf**. В этом случае в параметры некоторых методов не надо добавлять параметр типа **VMf.**

Структура struct VMTime для хранения результатов сравнения времени вычислений в разных режимах. Структура содержит открытые свойства и методы:

- свойство типа **VMGrid** для параметров сетки;
- свойство типа **VMf** с информацией о том, для какой функции выполнялось тестирование (если в **VMGrid** нет свойства типа **VMf**);
- свойства для времени вычисления в режимах точности **VML_HA** и **WML_EP** и времени вычисления значений функции в коде C++ без использования методов из библиотеки **Intel MKL** (можно определить одно или несколько свойств, которые возвращают массив и/или отдельные значения для каждого режима);
- два свойства типа **double** (или одно свойство типа **double[]**) для коэффициентов отношения времени вычислений в режимах **WML_EP** и **WML_HA** к времени вычислений в коде C++ без использования **Intel MKL**;
- версию метода **string ToString(),** который возвращает строку с данными структуры **VMTime.**

Структура struct VMAccuracy для хранения результатов сравнения точности при вычислениях в режимах **VML_HA** и **WML_EP** с открытыми свойствами и методами

- свойство типа **VMGrid** для параметров сетки;
- свойство типа **VMf** с информацией о том, для какой функции выполнялось тестирование (если в **VMGrid** нет свойства типа **VMf**);
- свойство для максимального значения модуля разности значений, вычисленных в режимах **VML_HA** и **WML_EP**;
- свойство(а) для значения аргумента функции, при котором максимально отличаются значения функции, вычисленные в режимах **VML_HA** и **WML_EP** (для одного из значений аргумента, если таких значений несколько), и соответствующие значения функции в режимах **VML_HA** и **WML_EP**;
- версию метода **string ToString(),** который возвращает строку с данными структуры **VMAccuracy.**

Класс VMBenchmark содержит

- свойство типа **ObservableCollection<VMTime>** для хранения результатов сравнения времени;
- свойство типа **ObservableCollection<VMAccuracy>** для хранения результатов сравнения точности вычисления функций;
- метод **AddVMTime** с параметрами типа **VMf** и **VMGrid** (или с одним параметром **VMGrid**, если **VMGrid** содержит свойство типа **VMf)**; в этом методе выполняются

вычисления, создается и добавляется в коллекцию **ObservableCollection<VMTime>** новый элемент **VMTime**;

- метод **AddVMAccuracy** с параметрами типа **VMf** и **VMGrid** (или с одним параметром **VMGrid)**; в этом методе выполняются вычисления, создается и добавляется в коллекцию **ObservableCollection<VMAccuracy>** новый элемент **VMAccuracy**;
- два свойства типа **double** (или одно свойство типа **double[]**), которые возвращают минимальные значение коэффициентов отношения времени вычислений в режимах **WML_EP** и **WML_HA** к времени вычислений без использования **Intel MKL** (среди всех элементов коллекции **ObservableCollection<VMTime>**);

Класс ViewData

В проекте **WPF** надо определить класс **ViewData,** который содержит свойства для привязки данных объекта **VMBenchmark** к элементам управления приложения**.** Объект типа **ViewData** является полем (или свойством) в классе главного окна приложения **WPF**. Класс **ViewData** содержит поле и/или открытое свойство типа **VMBenchmark**. Все операции приложения с данными **VMBenchmark** происходят через вызовы методов или свойств класса **ViewData.**

Класс **ViewData** содержит открытые

- свойство типа **VMBenchmark**;
- метод **AddVMTime** с параметрами типа **VMf** и **VMGrid** (или с одним параметром **VMGrid**, если **VMGrid** содержит свойство типа **VMf)**;
- метод **AddVMAccuracy** с параметрами типа **VMf** и **VMGrid** (или с одним параметром **VMGrid)**;
- метод **bool Save (string filename)**, в котором данные **VMBenchmark** сохраняются в файл с именем **filename**;
- экземплярный метод **bool Load (string filename)** или статический метод **bool Load (string filename, ref VMBenchmark vmb)**, в котором восстанавливаются данные из файла **filename;**
- свойство типа **bool** для хранения информации о том, что пользователь изменил данные объекта **VMBenchmark**.

Метод **void Save(string filename)**

- сохраняет данные объекта **VMBenchmark** в файл с именем **filename**;
- если файл с именем **filename** существует, приложение его перезаписывает; если такого файла нет, приложение его создает;
- метод бросает исключение, если в процессе сохранения данных или при создании/открытии файла произошла ошибка; если метод бросил исключение, пользователю надо сообщить информацию о причине исключения с помощью окна сообщений **System.Windows.MessageBox**;
- независимо от того, как завершилась операция, все файловые потоки должны быть закрыты в блоке **finally**.

Метод **void Load(string filename)**

- восстанавливает данные объекта **VMBenchmark** из файла с именем **filename**;
- метод бросает исключение, если при открытии файла или в процессе восстановления данных произошла ошибка; если метод бросил исключение, пользователю надо сообщить информацию о причине исключения с помощью окна сообщений **System.Windows.MessageBox**;
- независимо от того, как завершилось восстановление данных, все файловые потоки должны быть закрыты в блоке **finally**.

Для сохранения/восстановления данных в файле можно использовать любой из способов:

- сериализация с помощью **BinaryFormatter**;
- **JSON**-сериализация;
- сохранение/восстановление данных в текстовом формате с использованием **StreamWriter/StreamReader**;
- сохранение/восстановление данных в бинарном формате с использованием **BinaryWriter/BinaryReader**.

Пользовательский интерфейс программы

Главное окно приложения содержит меню с элементами

- **File** (с элементами **New**, **Open**, **Save**);
- **Edit** (с элементами **Add VMTime**, **Add VMAccuracy**).

Реакция приложения на выбор пользователем элементов меню File:

New – создается новый объект **VMBenchmark.**

Open – пользователь выбирает имя файла в стандартном диалоге **Microsoft.Win32**.**OpenFileDialog**. Если пользователь сделал выбор (закрыл диалог кнопкой **Open**) выполняется чтение данных в **VMBenchmark**.

Save – пользователь выбирает имя файла в стандартном диалоге **Microsoft.Win32**.**SaveFileDialog**. Если пользователь сделал выбор (закрыл диалог кнопкой **Save**), данные **VMBenchmark** сохраняются в файл с именем, который выбрал пользователь.

Если перед выбором элементов меню **New** или **Open** или перед выходом из приложения пользователь изменил **VMBenchmark** и не сохранил его в файле, он получает предупреждение о том, что данные будут потеряны**.** Предупреждение выводится с помощью окна сообщений **System.Windows.MessageBox.** Пользователю предлагается выбрать − сохранить в файле измененные данные или выполнить соответствующую операцию без сохранения результатов. Если пользователь выбрал сохранение данных, то вызывается стандартный диалог **Microsoft.Win32**.**SaveFileDialog** для выбора имени файла, в котором будут сохранены данные объекта **VMBenchmark**.

При **завершении работы приложения** проверку, что пользователь сохранил в файле измененные данные объекта **VMBenchmark**, надо выполнить обработчике события **Closed** главного окна приложения.

Реакция приложения на выбор пользователем элементов меню Edit

- **Add VMTime** – в коллекцию **ObservableCollection<VMTime>** добавляется новый элемент **VMTime**;
- **Add VMAccuracy** – в коллекцию **ObservableCollection<VMAccuracy>** добавляется элемент новый **VMAccuracy**;

Главное окно приложения содержит следующие элементы управления, в которые пользователь вводит данные для **VMGrid:**

- **ComboBox** для выбора функции, для которой выполняются вычисления;
- **TextBox** для ввода параметров сетки; все три значения вводятся в один элемент управления через разделитель; надо вывести информацию, в каком порядке надо вводить данные и какой разделитель используется.

Элемент управления **TextBox** надо связать с соответствующим свойством класса **ViewData** с помощью механизма привязки. В привязке надо использовать пользовательский преобразователь типа.

Главное окно приложения содержит следующие элементы для вывода информации**:**

- **ListBox** для вывода элементов коллекции **ObservableCollection<VMTime>;**
- **TextBlock** для вывода дополнительной информации об элементе, который пользователь выбрал в **ListBox** с коллекцией элементов **VMTime;**
- **ListBox** для вывода элементов коллекции **ObservableCollection<VMAccuracy>**;
- элемент(ы) **TextBlock** для вывода дополнительной информации об элементе, который пользователь выбрал в **ListBox** с коллекцией элементов **VMAccuracy;**
- **TextBlock** с информацией о том, что в процессе работы приложения пользователь изменил данные объекта **VMBenchmark**;
- элемент(ы) **TextBlock** с информацией о минимальном и максимальном значениях коэффициента отношения времени вычисления в режиме **VML_HA** к времени без использования **Intel MKL**.

Все перечисленные выше элементы управления надо связать со свойствами классов **ViewData** и **VMBenchmark** с помощью механизма привязки.

В главном окне приложения не должно быть обработчиков события **SelectionChanged** для элементов управления **ComboBox** и **ListBox**.

Привязку можно реализовать разными способами. Один из возможных способов − свойству **DataContext** главного окна приложения присвоить ссылку на объект **ViewData** и использовать свойство **DataContext** главного окна приложения как источник данных в привязках к свойствам класса **ViewData**.

Если все элементы управления, связанные с информацией из **ObservableCollection<VMTime>** разместить на одной панели, например, в элементе **Grid**, и свойству **DataContext** этой панели присвоить ссылку на объект **ObservableCollection<VMTime>,** то свойство **DataContext** панели можно использовать как источник данных в привязках к свойствам **ObservableCollection<VMTime>** или его

элементов. То же самое можно сделать для коллекции **ObservableCollection<VMAccuracy>**.

Вывод коллекции ObservableCollection<VMTime>

Все элементы коллекции **ObservableCollection<VMTime>** выводятся в элемент управления **ListBox**. Для каждого элемента выводятся имя функции и параметры сетки. При выводе используется шаблон данных **DataTemplate**. Шаблон данных **DataTemplate** надо определить в объектных ресурсах приложения.

Если шаблон **DataTemplate** содержит один элемент управления **TextBlock**, то для него в привязке надо использовать пользовательский преобразователь типа, который создает строку, содержащую необходимую информацию.

Можно определить шаблон **DataTemplate**, который содержит несколько элементов управления **TextBlock**, и для каждого элемента управления сделать привязку к соответствующему свойству из типа **VMTime**.

Для элемента, который пользователь выбрал в элементе управления **ListBox**, выводится дополнительная информация в элемент(ы) **TextBlock** – коэффициенты отношения времени в режимах **VML_HA** и **WML_EP** к времени вычислений без использования **Intel MKL.** В привязке к свойству(ам), определенному в типе **VMTime,** необходимо использовать пользовательский преобразователь типа, чтобы значение типа **double** выводилось с небольшим числом знаков после запятой.

Вывод коллекции ObservableCollection<VMAccuracy>

Все элементы коллекции **ObservableCollection<VMAccuracy>** выводятся в элемент управления **ListBox**. Для каждого элемента выводятся имя функции и параметры сетки. При выводе используется шаблон данных **DataTemplate**.

Для элемента, который пользователь выбрал в элементе **ListBox**, в элемент(ы) управления **TextBlock** выводится дополнительная информация – значение аргумента, при котором максимально отличаются значения функции, вычисленные в режимах **WML_EP** и **WML_HA,** и соответствующие значения функции. Можно выводить все данные в один элемент **TextBlock** и в привязке использовать пользовательский преобразователь типа. Можно для каждого свойства использовать отдельный **TextBlock**. Данные следует выводить со всеми знаками после запятой.

Обновление элементов управления при изменении коллекций VMBenchmark

При изменении коллекций **ObservableCollection<VMTime>** и **ObservableCollection<VMAccuracy>** (добавление элементов) должна обновляться информация

- в элементах **ListBox** с коллекциями;
- в элементе **TextBlock** с информацией о том, что пользователь изменил коллекцию;
- в элементах **TextBlock** с минимальным и максимальным значениями коэффициента отношения времени вычислений в режиме **VML_HA** к времени вычислений без использования **Intel MKL**.

Обновление данных в элементах управления **ListBox** происходит автоматически, так как универсальная коллекция **ObservableCollection<T>** сообщает о том, что в коллекции произошли изменения, с помощью события **CollectionChanged**. **WPF** обрабатывает событие **CollectionChanged,** если коллекция или ее представление используется как источник данных в привязке, и обновляет данные в элементах управления.

Для того, чтобы обновлялись элементы управления **TextBlock,** к свойствам которого выполнена привязка, должен быть реализован интерфейс **System.ComponentModel.INotifyPropertyChanged** в типах, которые используются как источник данных в привязке. Типы должны сообщать об изменении значений своих свойств с помощью события **PropertyChanged.**

Интерфейс **INotifyPropertyChanged** надо реализовать в классе **ViewData**. В конструкторе класса **ViewData** надо подписаться на события **CollectionChanged** от коллекций **ObservableCollection<VMTime>** и **ObservableCollection<VMAccuracy>** класса **VMBenchmark**. Тогда при каждом изменении коллекции будет вызываться обработчик этих событий (достаточно одного общего обработчика). В этом обработчике надо бросить события **PropertyChanged** для свойств, привязанных к элементам управления **TextBlock.**

Обработка исключений

Все исключения, которые могут возникать при обработке некорректного ввода пользователя, должны обрабатываться приложением.

Приложение должно оставаться в рабочем состоянии до тех пор, пока пользователь не закроет главное окно приложения.

Срок сдачи лабораторной работы

14 марта группы 306, 309, 341/2

15 марта группы 301, 302