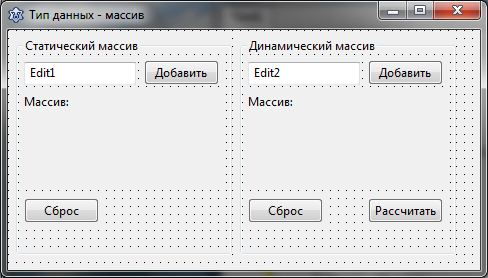
Лабораторная работа №4.



Тема: Тип данных - массив.

Задача: Разработать приложение, работающее с двумя массивами целых чисел (фиксированного размера в 10 элементов («статический») и переменного размера («динамический»)), позволяющее осуществлять поочередный независимый ввод элементов данных в эти массивы и выполняющее с заполненными массивами действия согласно варианту задания.

Массив представляет собой, по сути, одну из простейших составных структур данных. Массив - это просто некоторый набор (в современном программировании все наборы данных любой внутренней структуры называются коллекциями; таким образом, массив - простейший вид коллекции) однотипных элементов, записанных последовательно друг за другом в памяти эвм. Именно эта последовательность элементов, отстоящих друг от друга на одинаковое расстояние, равное размеру одного элемента, и определяет основные свойства массива как структуры данных.

Базовая операция, которую реализует массив — это доступ к элементу по его индексу. Каким образом она осуществляется? Для доступа к ячейке памяти программе нужно знать ее адрес. При этом компилятор запоминает все значения адресов всех переменных программы и вставляет их в необходимые участки машинного кода программы. Но современные программы оперируют большими объемами данных, поэтому хранить все множество адресов используемых ячеек памяти неразумно — оно будет сравнимо по размеру со всеми переменными в совокупности. В случае массива эта проблема решается очень просто — программа хранит только адрес начала массива, а адрес конкретного элемента вычисляется исходя из адреса массива, размера и индекса элемента. Для этой операции в современных процессорах даже есть специальная команда. В случае динамических массивов все происходит точно также, за исключением того, что адрес начала массива хранится не в коде программы, а в специальной переменной-указателе.

Массивы предназначены для хранения большого количества (больше одного, но как правило, целесообразность использования массивов начинается с 3-х) элементов, одинаковых по своему типу. Здесь следует отметить, что объект, несмотря на возможное разнообразие классов, это один тип. Также не рассматриваются вариантные типы, которые могут хранить различные по типу значения, прозрачно для программистов преобразующие их из одного представления в другое.

На практике основное преимущество массивов перед набором переменных заключается в возможности языка программирования организовать цикличную обработку его элементов. Например, есть массив из 10 элементов типа int:

…

int[] M = new int[10];

int count = 0;

for (int i = 0; i < 10; i++)

if (M[i] > 0)

count++;

…

В данном фрагменте кода мы циклически перебираем массив, подсчитывая количество элементов, больших 0.

Однако, этот набор элементов необходимо ввести пользователю тем или иным способом. На практике чаще всего используется ввод-вывод массивов с использованием файлов данных, т.к. вводить сотни тысяч значений пользователю неудобно и это занимает большое время. Но поскольку в данной работе максимальное количество значений в массиве ограничено десятками, мы будем использовать интерактивный ввод.

Для начала решим проблему ввода значений в статический массив. Как минимум, нам нужно знать, сколько значений уже ввел пользователь. Часто для этого можно применять специальные «флаговые» значения элементов, означающие, что соответствующий элемент пока не заполнен. Но в нашей задаче все множество значений типа Integer входит в область определения, поэтому данный подход неприменим. Заведем для этого вспомогательную переменную, записывая в нее количество введенных значений. Одновременно, легко заметить, что из этого значения легко получить индекс следующего вводимого элемента.

Запишем в глобальную область видимости:

int[] m = new int[10]; // «статический» массив

int inputCount = 0; // Счетчик элементов; это не просто 0, это первый из возможных индексов для m

Использовать эту переменную мы будем следующим образом: когда пользователь вводит значение, мы вычисляем индекс вводимого элемента, помещаем в массив на это место введенное число и увеличиваем количество введенных элементов на 1. В коде:

…

M[inputCount] = A; // A – число, введенное пользователем

inputCount++; // Инкремент, математически эквивалентно inputCount = inputCount + 1;

…

По завершению ввода, нам необходимо вычислить результат. Проще всего проверить значение переменной хранящей количество введенных элементов, и если это число равно (или больше) размеру массива, произвести вычисления. После вычислений необходимо сбросить состояние этого счетчика.

…

m[inputCount] = A; // A – число, введенное пользователем

inputCount++; // Инкремент, математически эквивалентно A := A + 1;

if (inputCount >= 10)

{

// вычисляем…

…

inputCount = 0;

};

…

Обратите внимание, что элементы самого массива не обнуляются — они будут в обязательном порядке перезаписаны в процессе ввода, и сбрасывать им значения не обязательно. Соответственно, сброс текущего состояния и будет заключаться, в основном, в сбросе счетчика введенных значений.

В случае с динамическим массивом все несколько сложнее. При добавлении каждого элемента необходимо выделять память специально, с помощью процедуры Array<T>.Resize() или пересоздавая массив каждый раз заново. При этом, хотя использовать внешнюю переменную размера массива и возможно, рациональнее воспользоваться методом Length, возвращающим текущую длину массива. На всякий случай, следует напомнить, что индексация всех массивов всегда начинается с 0.

В глобальной области видимости:

int[] md = {}; // Создаем пустой массив

В теле метода (обработка нажатия кнопки):

…

int[] new\_md = new int[md.Length+1];

md.CopyTo(new\_md, 0); // Копируем старый массив в новый

md = new\_md;

md[md.Length-1] = A;

…

В данном случае мы каждый раз пересоздаем массив, увеличивая его размер и копируя старое содержимое в новый массив. Точно также работает и метод Array<T>.Resize. Несмотря на кажущуюся сложность такой код всегда присутствует при работе с динамическими массивами. В данном случае он не очень удобен, но язык C# и не был рассчитан на активную работу с массивами. Для работы с наборами данных в нем предусмотрены более высокоуровневые коллекции с удобным интерфейсом, которые мы изучим позднее.

Для обнуления введенного массива просто присвоим переменной md значение пустого массива:

…

md = new int[0];

…

Одновременно с вводом данных в массив, необходимо осуществлять их вывод в метку на форме, чтобы пользователь мог видеть введенные данные и проверить корректность их ввода. Для обеспечения наилучшего вида рекомендуется установить свойство AutoSize у метки в false и увеличить ее размер по горизонтали до ширины группировочной панели, а по вертикали до нескольких (4-7) строк. Это позволит пользователю видеть ввод пары десятков больших по количеству цифр значений. При вводе значения и выводе его в метку лучшим выходом будет применение двойного преобразования — из строки в число и обратно из числа в строку:

void button1\_Click(object sender, EventArgs e)

{

int A;

if (int.TryParse(textBox1.Text, out A)

{

… // Добавляем в массив

label1.Text += A.ToString()+", "

}

}

В принципе, можно напрямую использовать пользовательский ввод, прибавляя строку EditN.Text, но в некоторых случаях это может выглядеть неопрятно, например, при вводе пользователем чисел 0xBABE или 0000001.

Задания

Вариант 1

1. Вычислить сумму элементов массива
2. Вычислить сумму 3-х максимальных элементов массива
3. Вычислить произведение элементов с четными индексами
4. Вычислить сумму элементов, больших среднего

Вариант 2

1. Вычислить произведение элементов
2. Вычислить произведение 2-х максимальных четных элементов
3. Вычислить сумму элементов с четными индексами
4. Подсчитать количество элементов, меньших среднего отрицательного

Вариант 3

1. Вычислить сумму четных элементов
2. Вычислить сумму 3-х элементов, ближайших к 0, но не равных ему
3. Вычислить произведение элементов с индексами, являющимися степенью 2
4. Вычислить произведение элементов, больших среднего по модулю

Вариант 4

1. Вычислить произведение модулей ненулевых элементов
2. Подсчитать количество элементов, меньших среднего по модулю
3. Вычислить сумму четных элементов с нечетными индексами
4. Подсчитать количество элементов, меньших половины максимального