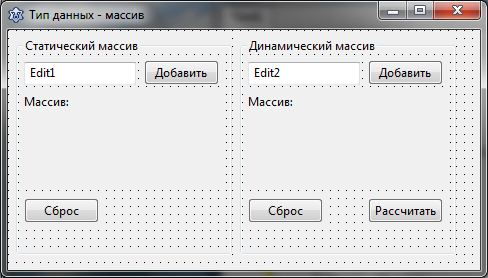
Лабораторная работа №4.



Тема: Тип данных - массив.

Задача: Разработать приложение, работающее с двумя массивами целых чисел (статическим размером 10 элементов и динамическим), 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 Integer:

var

M: array[0..9] of Integer;

i, Count: Integer;

begin

…

Count := 0;

For I := 0 to 9 do

If M[i] > 0 then

Inc(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.

Var

M: array[0..9] of Integer;

InputCount: Integer = 0; // 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

Inc(A); // Инкремент, математически эквивалентно A := A + 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

Inc(A); // Инкремент, математически эквивалентно A := A + 1;

If (A >= 10) then

Begin

// вычисляем…

…

A := 0;

End;

…

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. Для каждого элемента необходимо выделять память специально, с помощью процедуры SetLength. При этом, хотя использовать внешнюю переменную размера массива и возможно, рациональнее воспользоваться функцией Length, возвращающей текущую длину массива. На всякий случай, следует напомнить, что индексация динамических массивов всегда начинается с 0.

Var

MD: array of Integer;

…

Procedure …

Var

Len: Integer;

A: Integer;

Begin

… // ввод значения

Len := Length(MD);

SetLength(MD, Len+1);

MD[Len] := A;

…

End;

В данном случае мы используем локальную переменную, в которую временно помещаем значение. Это является хорошим тоном, повышает читаемость кода и несколько увеличивает быстродействие по сравнению с 3 вызовами функции Length. Одновременно мы избавляемся от ненужной глобальной переменной. Соответственно, в данном случае сброс будет заключаться в простой установке размера массива в 0:

…

SetLength(MD, 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, свойство WordWarp в true, 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:

Procedure ButtonNClick;

Var

A: Integer;

…

Begin

If TryStrToInt(EditN.Text, A) then

Begin

… // добавляем в массив

LabelN.Caption := LabelN.Caption + IntToStr(A) + ‘, ‘;

…

End;

End;

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

При удалении элемента из динамического массива необходимо сместить все следующие за ним элементы, поскольку при уменьшении размера массива будет отсечен последний элемент. Это можно сделать с помощью цикла:

var MD: array of Integer;

ToDelete, I, Len: Integer;

begin

… // Определение индекса удаляемого элемента

Len := Length(MD);

for i := ToDelete to Len – 2 do

MD[i] := MD[i+1];

SetLength(MD, Len-1);

end;

Можно заметить, что при удалении элемента из динамического массива таким способом присутствуют значительные накладные расходы на перезапись существенной части массива. Это важно при большом объеме данных задачи, поэтому для хранения очень больших размеров данных динамические массивы не являются достаточно эффективными — там используются другие, более сложные структуры данных (связные списки или даже списки массивов). Этого можно избежать, не смещая все элементы на 1 позицию, а перезаписав удаленный элемент последним. Данный подход не имеет таких больших накладных расходов, но в большинстве задач разрушает важные структурные свойства массива (такие как сортировка, порядок ввода элементов и пр.) и поэтому не применяется, если порядок следования элементов имеет значение в контексте данной задачи.

Следует также отметить особенность реализации цикла for в языке Pascal — а именно, что граничные значения цикла вычисляются один раз — перед его выполнением. Таким образом, в теле цикла не следует предпринимать действий, меняющих граничные условия — это может привести к ошибкам.

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