Лекция 6 (продолжение)

Динамические одномерные массивы

6.3 Типы памяти

В C++ существует три типа распределения памяти: статическое, автоматическое и динамическое.

Статическая *память*, куда компилятор помещает на время выполнения программы *глобальные данные и данные, объявленные с ключевым словом* `static`. Вкратце смысл переменной, объявленной с ключевым словом `static` таков. Если переменная внутри блока объявляется как `static` (например, `static int k=0;`), то обращаться к такой переменной также можно только внутри блока, но существовать она будет до завершения всей программы, т.е. значение переменной будет сохраняться до следующего выполнения этого блока. Таким образом, такая переменная инициализируется нулем только один раз, при первом вызове функции. Если при работе функции такая переменная получала другое значение, то при повторном вызове той же самой функции используется уже ее измененное значение. То есть между вызовами функции статическая переменная сохраняет то значение, которое она имела при последнем вызове.

6.3 Типы памяти

Автоматическая *память* – стек (структура данных, которая работает по принципу LIFO «последним пришёл — первым ушёл» или, другими словами, добавлять и удалять значения в стеке можно только с одной и той же стороны), которая до выполнения программного кода автоматически выделяется *локальным данным* и *параметрам функций* и автоматически освобождается при выходе из блоков. Напомним, блок – это любой фрагмент кода, заключенный в фигурные скобки, в частности составной оператор, и любая переменная, объявленная внутри блока или в теле функции, является локальной в этом блоке, т.е. «живет» только в блоке, после выхода из блока ее значение теряется, и обратиться к такой переменной вне блока нельзя.

6.3 Типы памяти

Например, если в теле блока объявлялась локальная переменная как

```
int k=0;
```

то, как только работа блока завершалась, то сама переменная становилась недоступной, память и выделенная для этой переменной, очищалась. Соответственно, эта память становилась доступной к перераспределению для других переменных. При каждом новом вызове функции под такую переменную заново выделялась память, и она снова инициализировалась, как в данном примере, нулем.

6.3 Типы памяти

Динамическая *память* – иногда ее называют ***куча***, в которой требуемый блок памяти выделяется только по явному запросу программиста, используя операции **new** и освобождается программистом тоже явно – оператором **delete**.

С помощью оператора **new** можно динамически выделить память для любого объекта. Пример выделения памяти в куче для одиночного значения целого типа **int**:

```
Int *p=new int;
```

Оператор **new**, исходя из указанного типа, выделяет необходимое число (**sizeof(int)**) байт памяти и адрес начала этой памяти присваивает указателю **p**.

6.3 Типы памяти

Выделенную оператором **new** динамическую память необходимо явно освободить после того, как она уже не нужна следующим образом:

```
delete p;//для освобождения блока памяти,адрес которого
         //содержится в указателе p, C++ использует
         //информацию, сформированную в служебном блоке
p=0;     //после освобождения памяти в переменной p еще
         //сохраняется значение адреса уже
         //недействительного блока,
         //поэтому безопаснее такой указатель обнулить
```

6.3 Типы памяти

Если выделенный динамически блок памяти программист забыл освободить, то эта память будет освобождена операционной системой только после завершения приложения. Поэтому, если выделять память интенсивно и большими блоками (особенно под графические объекты), то даже виртуальная память может закончиться. Особенно внимательно надо относиться к динамическим объектам, которые создаются внутри функций. Если функция использует локальную переменную как указатель на память, выделенную с использованием `new`, то указатель будет уничтожен по окончании работы функции, но память останется выделенной и при этом недоступной из программы. Поэтому освобождение памяти после того, как она была использована и больше не требуется, является хорошим тоном, а зачастую и просто необходимо.

6.3 Типы памяти

В предыдущей лекции и сейчас мы рассматриваем традиционные указатели языка C/C++. В Visual C++ понятие указателя расширено, и используются указатели двух типов:

- регулируемые указатели;
- нерегулируемые указатели.

6.3 Типы памяти

Понятие «регулируемый указатель» появляется при создании приложений в режиме CLR. CLR является добавкой-расширением C++, введенной фирмой Microsoft, начиная с версии VC++ 2005. Когда мы создаем программы и работаем с программными объектами без режима CLR, мы должны сами заботиться об их размещении в памяти. Память для приложения вне режима CLR выделяется в так называемой «неуправляемой куче», в ней мы сами размещаем свои объекты и сами должны освобождать память, когда перестаем работать с объектом, иначе куча может переполниться и процесс выполнения приложения прервется. Рассмотренные нами указатели, обозначаемые символом *, как раз являются указателями на участки памяти в такой «неуправляемой куче». Это традиционные указатели языка C/C++ на объекты в нерегулируемом объеме памяти, выделяемой приложению.

6.3 Типы памяти

Приложение в режиме CLR, отличается от обычного тем, что его заготовка обеспечивает подключение к приложению специального системного пространства System, содержащего объекты, размещение в памяти, которых надо регулировать. Режим CLR работает уже с управляемой кучей памяти, в которой размещение объектов и её освобождение от них происходит под управлением среды. Для этого введено понятие «регулируемый указатель» – это тип указателя, который ссылается на объекты, расположенные в общей регулируемой куче памяти, предоставленной приложению в момент исполнения. Для таких указателей введено специальное обозначение: вместо символа * применяется символ ^.

6.3 Типы памяти

Мы уже видели при работе с элементами управления на форме, что регулируемые указатели автоматически создаются в обработчиках событий компонентов. В среде VC++ существует специальная утилита ***gcnew***, которая формирует экземпляр объекта, выделяя экземпляру некоторую память, и возвращает ссылку на этот экземпляр. С регулируемыми указателями мы будем работать позже.

6.4 Динамические одномерные массивы

Чтобы создать динамический одномерный массив, необходимо указать тип и размерность массива. Например,

```
int n = 20;
float *p = new float [n];
```

В этом фрагменте создается переменная-указатель **p** на **float**, в динамической памяти отводится непрерывная область для размещения 20 элементов вещественного типа, и адрес начала этой области записывается в указатель **p**.

В принципе, создавать динамически массивы имеет смысл, когда:

- размер массива вычисляется во время выполнения программы;
- размер массива может изменяться во время работы программы.

6.4 Динамические одномерные массивы

Для освобождения блока памяти, который выделялся под массив оператором **new**, используется также оператор **delete**, но с квадратными скобками, которые означают, что освобождается массив:

```
delete[] p;
```

Как уже было показано ранее, для освобождения памяти, занятой скалярными объектами, квадратные скобки не нужны, но при освобождении памяти, занимаемой массивом, квадратные скобки обязательны. Их использование подразумевает, что мы освобождаем память, используемую для всех элементов массива, одновременно.

6.4 Динамические одномерные массивы Пример 6.8

```
...
unsigned int n;// здесь будет количество элементов
//вычислили n
// int ar[n];//Ошибка:размерность должна быть константой
int* pn=new int[n];//динамически выделяем требуемый блок памяти
                   // по контексту вызова оператора new компилятор
                  // выделит n*sizeof(int) байтов
//Использование динамического массива
for (int i=0;i<n;i++)
  *(pn+i)=...;
...
// Пусть в процессе выполнения программы оказалось,
// что нужно увеличить размерность массива вдвое:
int* tmp=new int[2*n];// выделили новую память
//переписали старое содержимое в новую область:
for (int i=0;i<n;i++)
     *(tmp+i)= *(pn+i);
delete[] pn;//старую область памяти освободили
pn=tmp;      //перенаправили указатель pn на новую область памяти
// и продолжаем пользоваться динамическим массивом
// с помощью указателя pn
*(pn+i)=...;
```

6.4 Динамические одномерные массивы
Пример 6.8

```
...
//когда массив больше не нужен память надо освободить:
delete[] pn;//освобождение динамической памяти.После
            //освобожд. памяти этим адресом пользоваться нельзя!
pn=0;       //безопаснее обнулить указатель, чем оставить
            // в нем значение недействительного адреса
...
```

6.4 Динамические одномерные массивы

Пример 6.9

Написать функцию формирования вещественного динамического массива из n элементов случайными числами из диапазона [a;b].

```
// ввод случайными числами динамического массива
float* input(int n, int a,int b)
{
    float* mas=new float[n];//выделение памяти под n веществ. знач.
    Random^ rnd = gcnew Random; // Создание объекта типа Random.
    for (int i = 0; i < n; i++)
         //Генерация случайных веществ. значений от a до b.
         *(mas+i) = a+(b-a)*rnd ->NextDouble();
    return mas;//возвращается указатель на массив
}
```

Вызов функции:

```
float *mas1=input(n1,a1,b1);
```

6.4 Динамические одномерные массивы Пример 6.10

Разработать процедуру формирования динамического вещественного массива **y** из положительных элементов исходного вещественного массива **x**.

6.4 Динамические одномерные массивы Пример 6.10.

```
// создание динамического  массива из положительных
// элементов другого массива
float* Pr610(float* x, int m, int& n)
{
    n=0;
    //подсчет числа эл-тов нового массива
    for (int i = 0; i < m; i++)
        if(*(x+i)>0)n++;
    if (!n) return 0;//выход из функции и возврат 0,
                     //если новый массив НЕ создается
    //ЕСЛИ выхода не произошло, то создается новый массив
    float* y=new float[n];//выделение памяти под новый массив
    n=0;                  //снова обнуляем счетчик
    // запись положительных чисел в новый массив:
     for (int i = 0; i < m; i++)
        if(*(x+i)>0)
        {
            *(y+n)=*(x+i);
            n++;
        }
    return y;          //возврат указателя на новый массив
}
```

6.4 Динамические одномерные массивы Пример 6.10.

```
/ фрагмент событийной процедуры примера 6.10
{
    ...
    int m=10;                          // кол-во эл-тов в 1-ом массиве
    int n;                             // кол-во эл-тов в новом массиве
    float* mas1=input(m,-5,5);//создание массива из случайных
                                       //чисел [-5,5]
    output(mas1,m,listBox1); //вывод 1-го массива в listBox1
    float* mas2=Pr610(mas1,m,n);//вызов ф-ции создания нового
                                        // массива
    if (mas2)                           //если новый массив создан
    {
       output(mas2,n,listBox2);//вывод 2-го массива в listBox2
       delete[] mas2;                  //очищение памяти 2-го массива,
                                       //только если он создавался
    }
    else MessageBox::Show("В исходном массиве нет
                          положительных элементов");
    delete[] mas1;//очищение памяти исх. массива в любом случ.
}
```

6.5. Пример выполнения лабораторной работы 6

1. **Задание**. Создать динамический одномерный массив **mas1** из чисел, записанных в многострочном текстовом поле **textbox** формы. Если в этом массиве есть хотя бы одно простое число, то сформировать новый массив **mas2** из тех элементов массива **mas1,** которые являются простыми числами.

2. **Формализация задачи**. Подобная задача была решена в предыдущей лабораторной работе №5, но теперь требуется формировать динамические массивы. Из условия задания следует, что его выполнение распадается на следующие этапы:

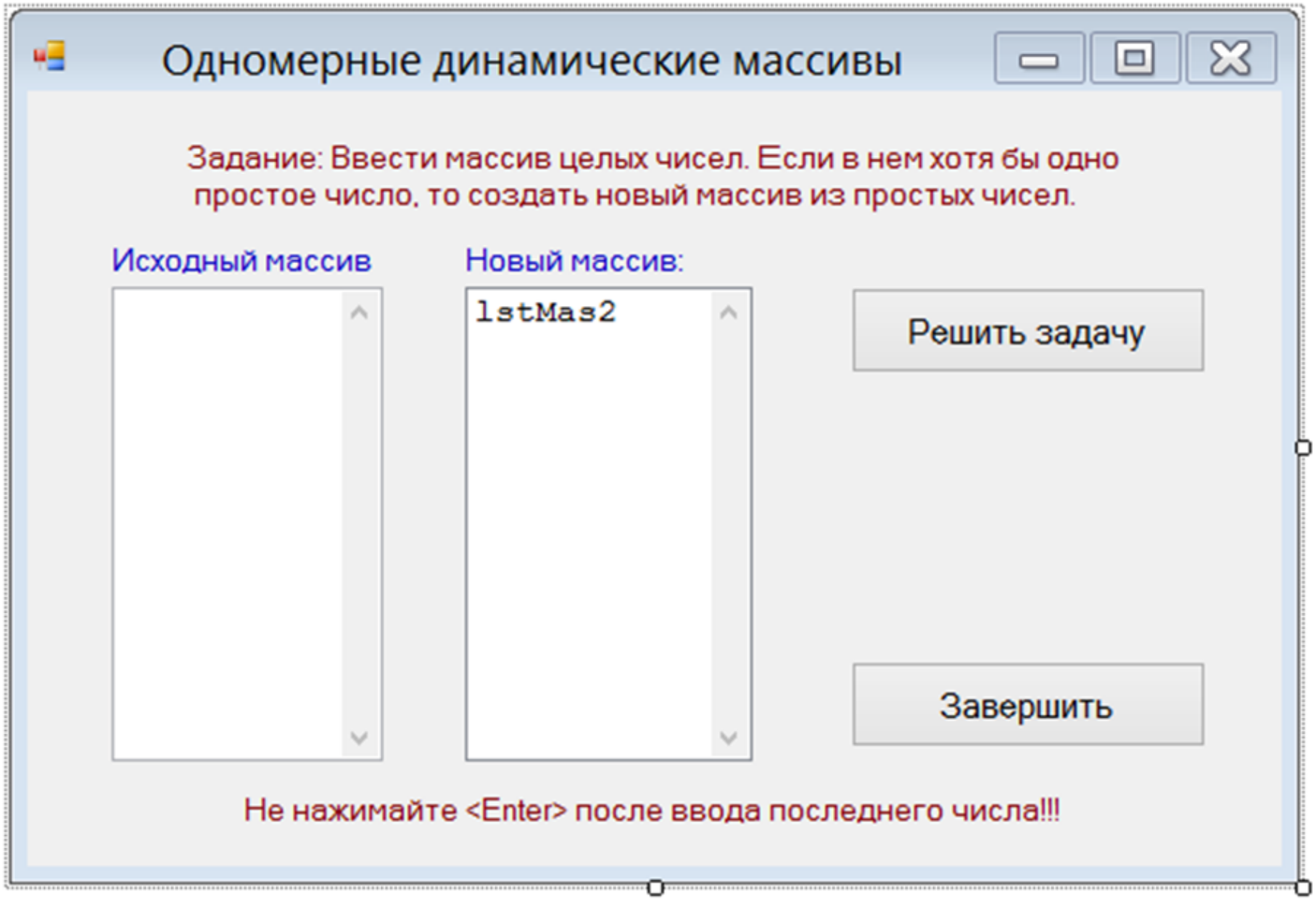
1) формирование массива **mas1** чтением чисел из текстового поля на форме;
2) анализ (проверка) есть ли в исходном массиве **mas1** хотя бы одно простое число, чтобы узнать, надо ли формировать новый массив;
3) формирование массива **mas2**;
4) вывод массива **mas2**.

6.5. Пример выполнения лабораторной работы 6

Для программной реализации задания создадим следующие процедуры:

1) функцию **`input()`** формирования массива чтением чисел из многострочного текстового поля на форме;
2) функцию **`output()`** для вывода одномерного массива целых чисел в заданный список (элемент управления ***ListBox***) формы;
3) функцию **`simple(),`** которая проверяет, является ли ее формальный параметр простым числом;
4) функцию **`analys()`**, которая анализирует, есть ли в массиве хотя бы одно простое число;
5) функцию **`task()`** формирования нового массива из простых чисел исходного массива.

6.5. Пример выполнения лабораторной работы 6



6.5. Пример выполнения лабораторной работы 6

```
// файл GetPut_din.cpp с ф-циями ввода и вывода
#include "stdafx.h"

// Функция ввода исходного массива из многострочного текстбокса
int*  input(int& n, TextBox^ TB)
{
    if ((n = TB->Lines->Length) == 0)// Если число строк в текстбоксе равно 0
        {
            MessageBox::Show("Заполните числами текстовое поле",
                        "Ошибка", MessageBoxButtons::OK,
                        MessageBoxIcon::Error);
            TB->Focus();
            return 0;           // признак ошибки ввода(массив не создан)
        }
    int* mas = new int[n];      // выделение памяти под n элементов массива
    for (int i = 0; i < n; i++) // Цикл заполнения массива.
        {
        String ^s = TB->Lines[i];       // i-я строка из текстбокса
        *(mas+i) = Convert::ToInt32(s); // конвертируется в целое
        }
    return mas;                         // возврат указателя на массив
}
```

6.5. Пример выполнения лабораторной работы 6

```
// Функция вывода n эл-тов массива в ListBox
void output(int* mas, int n, ListBox^ Lb)
{
    Lb->Items->Clear();
    for (int i = 0; i < n; i++)
        {
        String^ s = mas[i].ToString();// строка для записи в ListBox
        Lb->Items->Add(s);
        }
    if (!n) Lb->Items->Add("Массив пуст");
}
```

6.5. Пример выполнения лабораторной работы 6

```
// Файл task_din.cpp с функциями решения задачи
#include "stdafx.h"

// Функция определяет, является ли натуральное число простым
bool simple(int n)
{
    if (n < 2) return false;
    for(int i=2; i<=n/2; i++)            //перебор всех делителей
    //если остаток от деления равен нулю, то число непростое
        if (!(n%i)) return false;
    return true;
}

// Функция проверяет, есть ли в массиве хотя бы одно простое число
bool analys(int* a, int n)
{
    for (int i=0; i<n; i++)
        if (simple(*(a+i))) return true;// вызов функции проверки
                                        // простого числа
    return false;
}
```

6.5. Пример выполнения лабораторной работы 6

```
//ф-ция создания нового массива
int* task(int* mas, int n, int& k)
{
    k = 0;                          // счетчик чисел нового массива
    // вызов ф-ции анализа массива,
    //если она вернула 0, то массив не создается
    if (!analys(mas,n)) return 0;
    //===создание нового массива
    int* newmas = new int[n];    //выделение памяти под новый массив
    for (int i=0; i<n; i++)
        if (simple(*(mas+i)))  //если mas[i]- простое число
            {
                *(newmas+k) = *(mas+i);  //запись его в новый массив
                k++;   // увеличение счетчика чисел нового массива
            }
    return newmas;    //возврат указателя на созданный массив
}
```

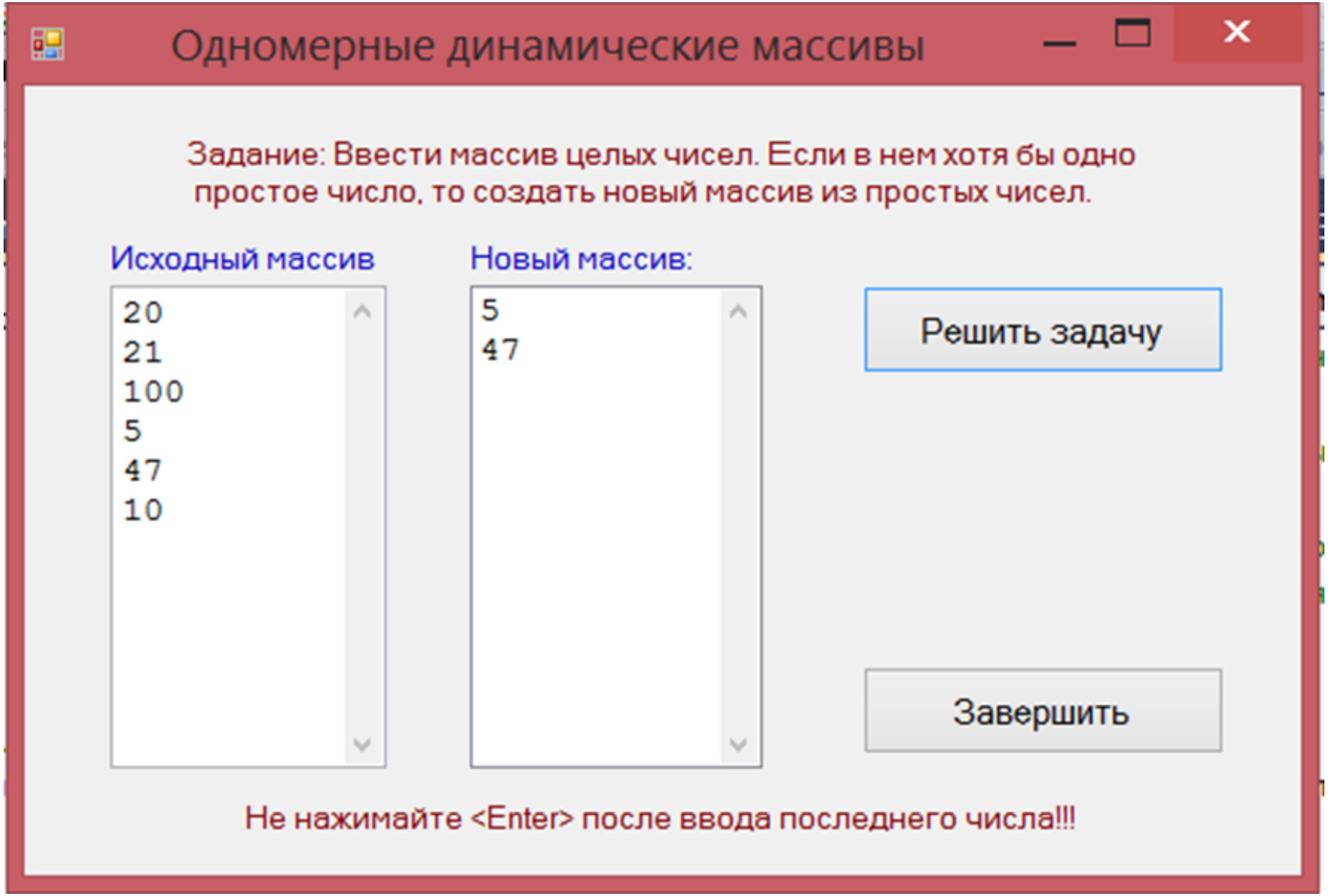
6.5. Пример выполнения лабораторной работы 6

```
// заголовочный файл task_din.h
using namespace System;
using namespace System::Windows::Forms;
// прототипы функций
int*  input(int&, TextBox^);
void output(int*, int, ListBox^);
int* task(int*, int, int&);
bool simple(int);
bool analys(int*, int);

// stdafx.h: включаемый файл для стандартных системных включаемых файлов
#pragma once

// TODO.
#include "task_din.h"
```

6.5. Пример выполнения лабораторной работы 6

```
//Событийная процедура кнопки "Решить задачу"
private: System::Void cmdTask_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
     lstMas2->Items->Clear();
     int n = 0;                          // число элементов в исходном массиве
     int k = 0;                          // число элементов в новом массиве
     int* mas = input(n, txtMas1);       // вызов ф-ции ввода массива
     if (!mas) return;                   // если исходный массив не создан
     int* newmas = task(mas, n, k);      // вызов ф-ции создания нового массива
     output(newmas, k, lstMas2);         // вывод нового массива в листбокс
     if (mas) delete[] mas;              // очистка памяти
     if (newmas) delete[] newmas;
 }

//Событийная процедура кнопки "Завершить"
private: System::Void cmdExit_Click(System::Object^  sender,
                                    System::EventArgs^  e)
 {
     this->Close();
 }
```

6.5. Пример выполнения лабораторной работы 6



6.5. Пример выполнения лабораторной работы 6