Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Ивановский государственный политехнический университет»
(ИВГПУ)

Кафедра ВПМСИТ УЦСГЕН

Основы работы в среде VC++ CLR

УЧЕБНОЕ ПОСОБИЕ
по дисциплинам
«Технологии программирования»
«Прикладное программирование в информационных системах»,
«Программная реализация средств дизайна»
для студентов дневной формы обучения
специальности 740100
«Информационные технологии в дизайне»

Иваново 2013

Учебное пособие предназначено для студентов, изучающих основы программирования в MS VC++ - ориентированной системе с управляемым кодом. Данное учебное пособие направлено так же на возможность приобретения некоторыми студентами базовых принципов решения задач по разработке программного обеспечения в системе VC++ CLR.

Учебное пособие является следующим этапом последовательности учебных материалов, направленных на изучения основ C++ и ознакомление с технологиями VC++.

Составитель:

Доцент кафедры ПМИТ, к.т.н., доцент Д. Д. Ветчинин

Научный редактор:

Заведующий кафедрой ПМИТ, д.т.н., профессор Н. А. Коробов

Введение

В этих учебных материалах Вам предлагается рассмотрение (с относительно невысокой, как обычно, вероятностью освоения) технологии программирования MS VC++ CLR. Пиар - менеджеры MS используют по отношению к этой технологии следующие понятия: «управляемый VC++»; «обновлённый синтаксис C++». Присвоение ими же этой программной технологии аббревиатуры CLR представляется понятным, но, не вполне корректным. MS VS.net, начиная с 2001 г. функционирует в среде framework.net, а разработанные проекты компилируются двухстадийно в подсистеме CLR (Common Langvige Runtime). Это, в частности, определяет платформенную независимость программных разработок.

Основные приёмы разработки приложений в IDE VC++ CLR будем рассматривать на примере программной реализации знакомого Вам приложения «Экзаменационный билет». Хотя это приложение имеет свидетельство о регистрации /1/, предлагаемый к изучению его реализационный вариант полноценными функциональными возможностями программного обеспечения данного назначения не обладает. Этот вариант создан именно для организации изучения основ программирования в среде VC++ CLR. При этом функциональная возможность организации крайне локализованного аттестационного мероприятия у рассматриваемой программы всё же имеется.

Полый исходный программный текст (code) программы «Экзаменационный билет» приведён в приложении. Представляется вероятным то, что на начинающих изучение VC++ CLR объём и состав данного кода первоначально может произвести шокирующее (если не отталкивающее) впечатление. Также представляется вероятным то, что для части студентов такое восприятие кода программы изменится в лучшую сторону после прохождения первого же этапа изучения, соответствующего первому этапу создания проекта VC++ CLR.

Исходя из вышеизложенного, Вам можно сделать следующие краткие резюмирующие выводы:

- предлагаемая учебная работа связана с одной из перспективных на настоящий момент технологий программирования с использованием «обновлённого синтаксиса C++» - VC++ CLR;

- эта технология программирования основана на принципах .net и полностью «параллельна» таким ветвям разработки ПО MS VS, как VC# и VB.net;

- для освоения учебного материала в качестве образца Вам предлагается пример эскизной работы профессионального разработчика прикладного ПО – локализованный учебный вариант приложения - Экзаменационный билет»;

- освоение учебного материала предлагается поэтапно. Во-первых, имеются в виду два этапа создания проекта. Во-вторых, этапы детализации проекта образца. Попробуйте повторить и/или сделать нечто своё.

1. Этап визуального проектирования

Этап «визуального проектирования» приложения в принципе одинаковов в большинстве IDE, например: VB, Delphy, C++Builder, C#, диалоговые окна C++ MFC и т.п. Этот этап программирования на настоящий момент должен быть Вами хорошо освоен. Следует только отметить разночтение в наименованиях элементов управления различных IDE MS.

Результат первого этапа создания приложения «экзаменационный билет» показан на рисунке 1. «Статически» на форме размещены четыре элемента управления и заданы свойства этих пяти объектов. Объекты, поименованные на рисунке 1 как «элементы runtime», определяются «динамически» - в ходе выполнения программы.

Исходный программный текст, сформированный системой в результате «визуального проектирования», представлен в приложении 1 строками 112 – 235. Определён этот блок кода директивами препроцессора #pragma region … #pragma endregion. Следует отметить, что в «параллельной» по стилю программирования системе C# этот блок кода «по умолчанию» скрыт от разработчика.

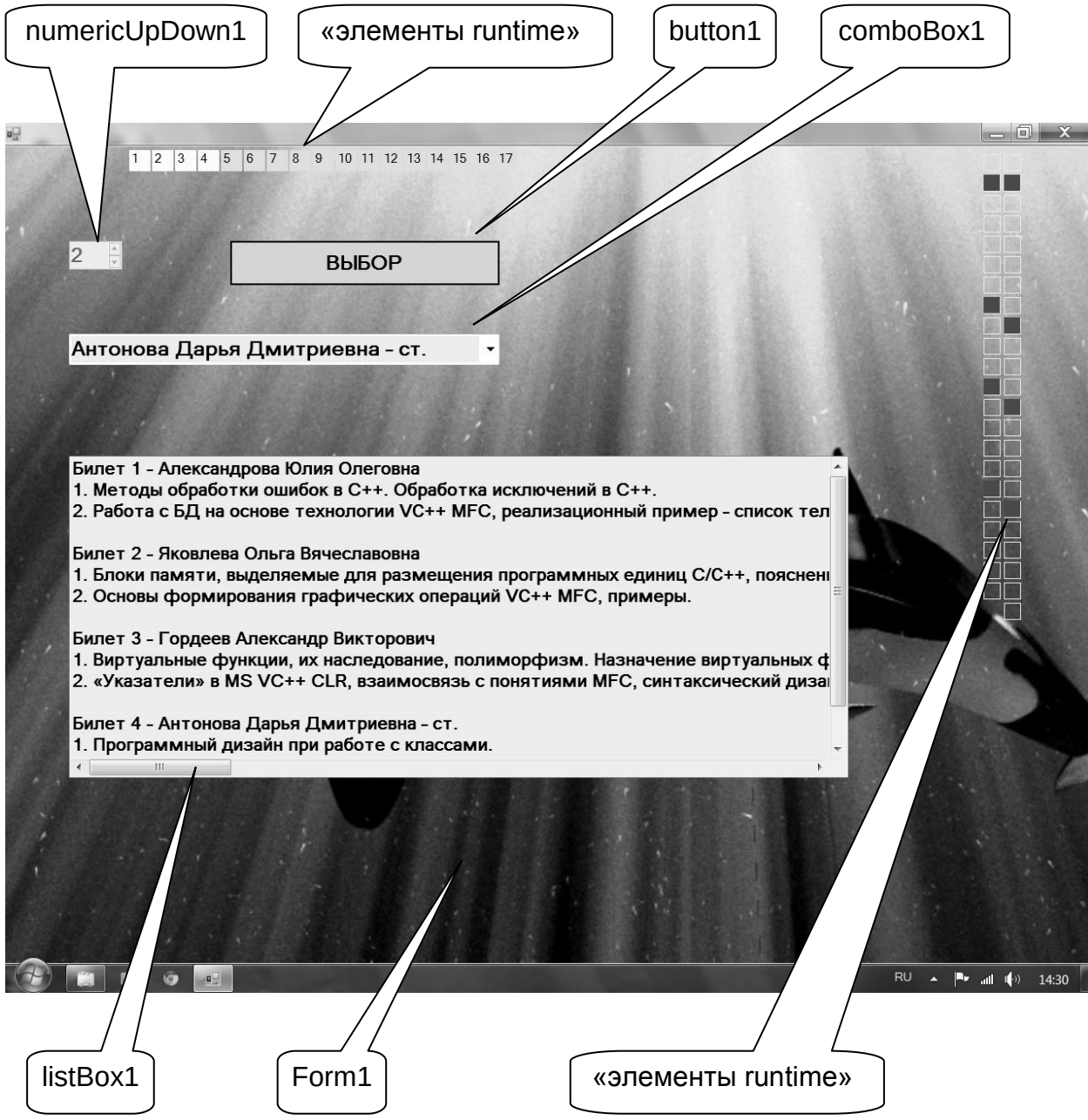

Рис.1. Интерфейс приложения «Экзаменационный билет»

2. Конструктор формы Form1

Конструктор формы реализован программными строками 25 – 85.

Строка 28 – создание «указателя» (точнее, хендлера /2/) с именем fr на объект класса StreamReader (поток чтения). Синтаксическая конструкция StreamReader^ fr является примером обновлённого (причём, в данном случае – дважды) MS синтаксиса C++ с соответствующей поддержкой программными подсистемами – организация управляемого кода. Почему имеет место замечание – дважды (по меньшей мере) обновлённый синтаксис C++? VC++ до версии 2008 вместо символа ^ использовал набор символов __gc * , т.е., рассматриваемая синтаксическая конструкция в предыдущих версиях VC++.net выглядит так: StreamReader__gc * fr . Впрочем, аббревиатура gc использована перед традиционным для C символом указателя вроде бы осмысленно и, поэтому унаследована другим оператором управляемого VC++. Garbage collection (уборка мусора) видимо, подразумевает освобождение разработчика, использующего VC++.net, от забот с динамически выделяемой памятью. На практике – это и исключение необходимости использования оператора delete.

Строки 30 - 40 – этот участок кода задаёт динамическое выделение блока памяти для размещения информации, считанной из файла. Блок кода «охвачен» традиционным для C обработчиком исключений (try … catch) /3/.

Строки 32 - 34 – строкой определяется управляемое системой динамическое выделение памяти в «куче» (on the garbage collected heap) для размещения в этом объекте - блоке памяти конкретной информации. Информация конкретизируется использованием функции StreamReader с передачей ей нужных аргументов: 1-ый аргумент – имя текстового файла и детализация его локализации (в данном примере – директория Debug директории проекта); 2-ой аргумент – выбор «кодировки» текста (в данном примере – «кириллица» (1251) в «бинарном состоянии»). Результатом выполнения оператора строки 32 – 34 является размещение информации – список студенческой группы в блоке памяти «управляемой кучи», к которому можно обратиться через хендлер с именем fr.

СПРАВОЧНОЕ: **gcnew** creates an instance of a managed type (reference or value type) on the garbage collected heap. The result of the evaluation of a **gcnew** expression is a handle (^) to the type being created.

ВНИМАНИЕ: имена текстовых файлов в рассматриваемой программе заданы «жёстко», кроме того, эти файлы сформированы по

определённой схеме, например, файл – список студентов: 1 – ая строка это количество студентов в списке; соответствующее число последующих строк – список группы (приложение 1).

ДОПОЛНИТЕЛЬНО: т.е., если бы файл Stu.txt отсутствовал в заданном месте или имел бы другое имя, произошёл бы аварийный «жёсткий» останов пограммы («классический С» - terminate(), aborte()). Для организации «мягкого» выхода из аварийной ситуации (без закрытия программы) используем технологию С – «перехват исключений».

Строка 30 – задание try – блока для выброса исключения (реагирование на любую программную ошибку). Этим блоком «охвачена» операция файловая операция gcnew .

Строка 36 – задание catch – блока для «перехвата» выброшенного исключения. Аргумент функции catch() - Exception^ определяет «перехват» любого исключения, сгенерированного вышерасположенным блоком try . В данном catch – блоке выводится окно сообщения об отсутствии требуемого текстового файла с предложением «мягкого» завершения работы. Явно указанное имя хендлера в данном варианте очевидно не требуется.

Строка 43 – определение переменной-счётчика с инициализацией 0.

Строка 44 – присвоение значения переменной stu (определена в строке 88). Convert::ToInt16() – функция преобразования в данном примере к двухбайтному целому. В аргументе функции - fr->ReadLine(). Т.е., хендлер fr через оператор -> вызывает функцию объекта типа StreamReader - ReadLine() (чтение строки). Результат – считывание первой строки текстового блока (список студентов) и присвоение переменной stu двухбайтного целочисленного значения, соответствующего числу студентов в списке.

Строка 45 – переменная – хендлер ls (определён в строке 89) «устанавливается» на одномерный (String^) строковый массив, число строк которого задаётся равным числу студентов (stu). Блок памяти для массива размещается в garbage collected heap.

Строка 46 – хендлер openLS (определён в строке 98) «устанавливается на» одномерный (<int>) массив с размерностью stu, размещение блока данных массива, естественно, там же – в garbage collected heap .

Строка 48 – определение переменной int x1 с её инициализацией значением, установленным элементом управления – счётчиком при инициализации, т.е., numericUpDown1->Minimum = 1; (строка 79).

Строка 49 – определение переменных y1, lw, lh, dx с их инициализацией значениями в пикселях. Эти переменные используютс при работе работы с «динамически размещаемыми» элементами управлениия (строки 54 – 65) и, соответственно, представляют собой: координату y, ширину, высоту, изменяемую координату x.

Строка 51 – выделение garbage collected heap для одномерного массива элементов Label с размерностью stu, хендлер на массив определён в строке 102. Специфика – работа с элементами управления «в динамике» (в период runtime).

Строки 53 - 72 – этот блок кода предназначен для размещения вышеопределённых элементов Label в окне программы с определением их информационно-содержательной визуализации. Блок «охвачен» обработчиком исключений (try … catch), «срабатывающим» на любую ошибку. Т.о., если бы число строк в текстовом файле – список студентов не соответствовало количеству, заявленному в первой строке, случится выброс исключения (аналогия с блоком кода 30 – 40). Выброс исключения произойдёт и в случае несоответствия числа строк их списка заявленному числу в первой строке файла.

Строки 54 - 65 – «рабочий» цикл while «внутри» вышерассмотренного блока 53 - 72. Условие повторения цикла должно быть для Вас достаточно прозрачно читаемым – тело цикла повторяется для каждой строки файлового потока fr до тех пор, пока «истиной» является то, что поток «не исчерпал себя» (EndOfStream).

В теле цикла while строками 56 – 61 формируется дизайн элементов Label (визуализация количества студентов группы с последующим контролем явившихся на экзамен). Что конкретно задаётся в теле цикла?

Строка 56 – вывод элемента Label - lbl[i] на окно программы. Строка 57 – установка координат и размеров элементов Label.

Строка 59 – задание вывода на элемент Label текста – целое число (порядковый номер элемента), преобразованное функцией ToString() в тестовое отображение (первая строка игнорируется (i+1), т.к., она содержит информацию о количестве студентов, а не о студенте).

Строка 60 – вызов метода Add() для этих элементов управления – вывод на окно программы.

Строка 61 – изменение горизонтальной координаты для каждого следующего элемента.

Строка 63 – построчное считывание в массив ls строк файлового потока fr методом ReadLine().

Строка 64 – «заполнение» элемента comboBox1 списком студентов через вызов его свойства Items и соответствующего ему метода Add(). Постинкремент Add(ls[i++]) задаёт итерационность цикла.

Строки 67 - 72 – блок catch() - Exception^ в комментариях на настоящий момент прочтения учебного пособия нуждаться не должен.

Строка 74 – закрытие файлового потока для чтения.

Строка 75 – установка элемента comboBox1 (список студентов) на первую позицию в считанном списке студенческой группы.

Строки 78 - 80 – установка требуемых свойств элемента numericUpDown1 («жёсткое» задание числа экзаменационных вопросов в билете): минимум – 1; максимум -4.

Строки 83 - 84 – начальные шаги по варианту формирования «графики» (глава 8, страница 22, строка 407). Задаётся функция «с идентификатором программиста» drawPI() обработки системного события Paint /5/. «Пройдёмся» по синтаксису: this - указатель на Form1 ; this->Paint - обращение к событию на форме - «начали рисовать»; += gcnew - выделение garbage collected heap для работы с «графикой»; PaintEventHandler() - функция реализации хендлера на графический объект, имена базовых классов можно «увидеть слева» от имени PaintEventHandler . В аргументе функции PaintEventHandler() имеют место два аргумента: &Form1::drawPI - задействовано обращение к функции drawPI() , имеющей место в адресном пространстве формы Form1 ; this – «системный» указатель для хендлера на «графику».

Рекомендовали бы Вам работу с модулем графики отложить на этап попытки итогового осмысления базовых принципов работы изучаемой пограммы.

Строки 87 - 102 – под квалификатором private определено несколько программных объектов:

- static int nq – переменная для количества экзаменационных вопросов в билете (присвоение значения в строке 240);

- static int stu - переменная для хранения значения количества студентов в группе (строки 44, 45, 46, 51);

- array<String^>^ ls – хендлер на массив для списка студентов (строки 24, 63, 64);

- array<String^>^ tt - хендлер (строка 89) на «буферный» строковый массив для одного экзаменационного билета, инициирован в строке 278 количеством ячеек на единицу болшим числа вопросов в билете (этой «лишней» строкой определяется персонификация экзаменационного билета;

- array<String^>^ q - хендлер на «буферный» массив для блока экзаменационных вопросов (строка 334);

- int n – «буферная» переменная для числа вопросов в блоке (строка 332);

- String^ st - хендлер на строковый объект (определён в строке 90). Можно рассматривать как буферный объект для формирования строки одного из вопросов билета. Объект st заполняется значением строкового литерала в строке 370.

- array<String^>^ q1 – хендлер на массив вопросов 1-го блока (строки 245, 247, 250, 254, 293,);

- array<String^>^ q2 – хендлер на массив вопросов 2-го блока (строки 247, 250, 254);

- array<String^>^ q3 – хендлер на массив вопросов 3-го блока (строки 250, 254);

- array<String^>^ q4 – хендлер на массив вопросов 4-го блока (строка 254);

- int n1, n2, n3, n4 – предназначены для количества вопросов в блоках вопросов (строки 241 - 263);

- nn - переменная для значения наибольшего количества вопросов в одном из заданных блоков вопросов (строки 95, 360, 422);

- int static k = 1 – переменная-счётчик для нумерации студентов в порядке выбора ими билета (т.е., и для нумерации билетов). Определена как static поскольку используется различными программными модулями (строки 96, 402, 403, 371, 280, 283, 288, 315);

- static array<int>^ openLS - хендлер на одномерный массив int , в котором предполагается размещение индекса студента (индекс в списке) в порядке выбора студентами экзаменационных билетов. Т.е., например, студент с индексом 5, первым нажавший кнопку «Выбрать» будет «прописан» в ячейке массива по openLS с индексом 0, студент

с индексом 0 вторым нажавший кнопку «Выбрать», будет «прописан» в ячейке массива по openLS с индексом 1 (строки 281, 286, 402,). Массиив определён как static , поскольку используется различными программными модулями. Назначение массива – исключение повторного выбора билета студентом.

- static array<int,2>^ openQ - хендлер на двухмерный массив int (определён в строке 99), в ячеках которого размещаются индексы «выбранных рандомизацией» экзаменационных вопросов блоков вопросов. Индексы строк массива (k – 1) определяются номерами билетов, тождественными нумерации студентов в порядке выбора ими билетов. Индекс стобца – индекс блока вопросов. Массив предназначен для записи индексов выбранных вопросов в системной связи с номерами билетов, т.е., и с номерами студентов в порядке выбора билета. Массив предназначен для регистрации выбора билетов – при выборе билета учитываются студенты и полученные ими вопросы. Назначение – исключение повторов выбора вопросов. Массиив определён как static поскольку используется различными программными модулями (строки 265, 269, 368, 371);

- static array<int,3>^ openQG - - хендлер на 3-х мерный массив int со спецификатором static (определён в строке 100). Оператором строки 266 выделен блок памяти для этого массива (garbage collected heap) естественно, с конкретизацией размеров массива. Образно массив представляет собой 3 таблицы с числом столбцов nq (число вопросов в билете) и количеством строк nn (максимальное количество вопросов по самому большому блоку вопросов). Массив предназначен для хранения значений начальных координат графических элементов, матрица которых отображает процесс формирования экзаменационных билетов. Массив «набирает» «рабочие значения» операторами строк 429 - 431.

- array<System::Windows::Forms::Label^> ^lbl - хендлер на одномерный массив Label, предназначенного для индексной индикации списка студентов и их явки на экзамен;

3. Функция – обработчик события «задание числа вопросов в билете»

Сгенерировать заготовку подобной функции достаточно тривиально: окно Properties – вкладка Event; выбор необходимого события; раскрытие «загадочного» пустого списка – и почему-то «двойной клик». В результате «заготовка» тела функции имеет место в исходном коде, а Вы в её «пустом, редактируемом теле».

Объявление функции – строка 237, 238. Тело функции – блок строк 239 -273. Детализируем синтаксис объявления функции при её определении:

- конструкция private: System::Void numericUpDown1_ValueChanged в комментариях нуждаться не должна, последующий набор символов попробуем откомментировать:

- первый аргумент функции (System::Object^ sender). Sender – передатчик, отправитель информации (классика англ. яз.). В данной синтаксической конструкции sender является именем собственным программного объекта – хендлера на системный программный объект, генерирующий событие, которое обрабатывает функция;

- второй аргумент функции (System::EventArgs^ e). В некоторых источниках /4/ этот аргумент (e) поименован «экземпляром класса EventArgs – базового для объектов, содержащих информацию о событии». В /5/ EventArgs — это базовый класс для классов, содержащих данные о событии. В нашем понимании «обновлённого C++, т.е. - CLR» (полная «параллель» C# и VB.net) с использованием базовых концепций C++ аргумент e можно охарактеризовать следующим образом: e – программный объект (имя собственное MS, изменению в данном синтаксисе не подлежит) - хендлер класса EventArgs, необходимый для программной организации действий, зависящих от информации о событии.

Рассмотрим построково исходный программный текст тела функции.

Строка 240 – присвоение значения переменной (число экзаменационных вопросов в билете) nq (определена в строке 88). Получает значение, отдаваемое numericUpDown1, преобразованное в двухбайтное целое.

Строка 241 – первичная инициализация переменных n1, n2, n3, n4 (определены в строке 95) нулём.

Строки 242 - 258 – блок switch с аргументом nq (номер блока вопросов – количество вопросов в билете).

Поскольку в данном программном примере «жёстко» определена возможность формирования билета с количеством вопросов не более четырёх – блок switch здесь имеет четыре оператора выбора case (соответствие номеру вопроса в билете - номеру блока вопросов).

Для компактизации объёмов учебного материала ограничимся сравнительным анализом двух вариантов case, первого и второго:

case 1: (numericUpDown1 задано число вопросов в билете - 1). Вызывается функция fq() с передачей ей в качестве аргумента имени файла первого блока вопросов (имена фалов блоков вопросов заданы «жёстко», например первый из них q1.txt, расположены они в директории Debug проекта). Для понимания дальнейшей работы функции Void numericUpDown1_ValueChanged() необходимо изучить действия функции fq() (определена строками 323 – 357), их детальное рассмотрение имеет место в 4 – ой главе учебного пособия (страница 15). Прежде чем закончить изучение дальнейших действий фунции, являющейся предметом анализа данной главы (глава 3), целесообразным было бы прочтение с попыткой осмысления учебных материалов, предлагаемых главой 4.

После вызова функции fq(("\\q1.txt") выполняются операторы n1 = n; q1 = q;. Результат: в ячейке переменной n1 - число вопросов в первом блоке; в массиве по хендлеру q1 - вопросы 1-го блока (хендлеры q1, q2, q3, q4 определены в строках 92, 93). Оператор break - переход на строку 259.

case 2: (numericUpDown1 задано число вопросов в билете - 2). В этом случае функция fq() вызывается дважды с разными значениями аргумента (строки 347, 348). Результат: в ячейке переменной n1 - число вопросов в первом блоке; в массиве по хендлеру q1 - вопросы 1-го блока; в ячейке переменной n2 - число вопросов во втором блоке; в массиве по хендлеру q2 - вопросы 2-го блока. Оператор break - переход на строку 259.

Строка 260 – присвоение локальной переменной функции nn (определена в строке 95) первоначального значения, соответствующему числу вопросов в первом блоке.

Строки 261 - 263 – строки поиска максимального числа вопросов в блоках вопросов. Результат – присвоение значения максимального

количества вопросов, определённых в одном из блоков вопросов по сравнению с другими, переменной nn (она определена в строке 99).

Строка 265 – (хендлер openQ на двухмерный массив определён в строке 95), создание двухмерного массива int (математически, матрицы с числом строк nn - количество студентов, число столбцов nq – количество вопросов в билете), естественно размещение данных в garbage collected heap. Массив предназначен для сохранения информации, связывающей индекс студента в порядке выбора вопросов (номер билета) с индексами назначенных ему вопросов (заполнение массива значениями – оператором строки 371, строка 368 – условие исключения повтора, строка 265 – исключение повторов).

Строка 266 – (хендлер openQG на трёхмерный массив определён в строке 100), создание трёхмерного массива int (математически, 3d-матрица, каждая из трёх таблиц которой имеет число строк nn - количество студентов, число столбцов nq – количество вопросов в билете), размещение данных в garbage collected heap. Для размещения начальных координат графических элементов (глава 8).

Строки 268 - 269 – «цикл в цикле – двойное вложение». Назначение – первоначальное заполнение ячеек массива openQ одинаковыми числами (nn + 1 – индекс студента в списке, которого не может быть). Внешний цикл – выбор столбца, внутренний цикл – перебор строк.

Строка 271 – вызов метода Refresh() элемента numericUpDown1 - обновление всех данных.

4. Функция fq() – считывания вопросов из файлов

Строка 323 – void fq(String^ f) – задаётся определение функции. По «заголовку» определения «видно», что функция принимает один аргумент – хендлер на объект класса String и, наследуя «классический» C++ , синтаксис ^ f является локализованной в этой функции определением переменной – хендлера.

Строки 324 - 347 – тело определяемой функции.

Строка 325 – определение локального для функции fq() хендлера fr для работы с потоком чтения из файла.

Строки 326 - 355 – блок кода «охваченный» (try … catch) /3/, причём catch – блоки в данной реализации многовариантны. Строка 326 – «выброс» исключения при любой ошибке в работе с файловым потоком.

Строка 328 - 330 – переменная – хендлер fr (определена в строке 28) «устанавливается» на поток чтения одного из файлов (блоков) вопросов. Для размещения информации файлового потока в массиве String задаётся garbage collected heap. Все прочие характеристики строки рассмотрены в главе 2 (страница 5). Но, есть одна особенность: в строке 33 (StartupPath + "\\Stu.txt") манипулируем именем файла «жёстко»; в строке 39 – имя файла задаётся «помягче», с использованием переменной f - локальной переменной – хендлера на объект класса String. Последняя программная вариация могла бы подтолкнуть к мысли об одних из направлений профессионализации рассматриваемой программы и подобных ей программных заготовках.

Строка 332 – переменной n (определение в строке 90) присваивается значение, переданное файловым потоком fr по результатам считывания первой строки файла (число вопросов в блоке вопросов), как и прежде, преобразовано в двухбайтное целое.

Строка 333 – определение локальной переменной - счётчика i и инициализация его нулём.

Строка 334 – переменная - хендлер q (определён в строке 90) – одномерный массив String размерностью n (число вопросов в блоке) в garbage collected heap.

Строки 336 - 339 – цикл, реализующий построчную запись в массив по хендлеру q строк, считанных из файлового потока fr (один из файлов вопросов) с использованием метода ReadLine(). Цикл

выполняется до тех пор, пока не достигнут конец файла (потока) - EndOfStream.

Строки 341 - 347 – первый catch – блок, перехватывающий исключение – отсутствие любого из файлов экзаменационных вопросов.

Строки 349 - 355 – второй catch – блок, перехватывающий исключение –ошибка в структуре любого из файлов экзаменационных вопросов (вообще то, в аргументе этого catch() задан перехват любого исключения.

Строка 356 – закрытие файлового потока fr.

Т.о., функция fq() считывает строки вопросов из одного из подготовленных файлов вопросов (имя файла передаётся ей в качестве аргумента при вызове) и размещает их в строковом массиве, который можно рассматривать как буферный. После изучения этой главы можно возвращаться к материалам главы 3 (страница 12, строка 245 программы).

5. Функция – обработчик события «выбрать билет»

Строки 275 - 276 – заголовок определения функции (строки 277 - 320) обработки события button1_Click(). Аргументы функции рассмотрены ранее.

Строка 278 – переменная - хендлер tt (определён в строке 89) – одномерный массив String размерностью nq + 1 (nq - число вопросов в билете) в garbage collected heap. Назначение массива – формирование полного текстового содержимого экзаменационного билета.

Строка 280 – переменная int static k = 1 (строка 96), которая хранит номер выбора билета (номер билет – студент) определена и инициализирована значением 1 (первый выбор) при инициализации формы. Т.е., первое нажатие кнопки «Выбрать» вызовет выполнение условия k == 1 , в результате будет заблокирован numericUpDown1 (последующее изменение числа вопросов в билете становится невозможным) и выполняется цикл for (строка 281);

Строка 281 – цикл первоначального заполнения ячеек массива по хендлеру openLS (определён в строке 98) одинаковыми значениями

(stu - количество студентов) – заполнение заведомо невозможным индексом списка для организации последующей проверки повторов.

Строка 283 – если «вдруг» значение счётчика k превысило число студентов – безусловный переход «на метку» end (строка 318, строка 319 – обязательный «по правилам С» оператор, в данном случае – пустой) - фактическое блокирование возможности формирования «избыточных» билетов;

Строки 285 - 286 – цикл for «поиска» «порядковой» ячейки массива по openLS , в которой «уже, возможно, прописан» индекс студента с сформированным билетом. Цикл «запускается», если предыдущее условие (строка 283) - false . Итерационные условия цикла: инкремент с шагом 1 списка студентов по индексам, начиная с 0 до индекса, на 1 меньшего количества студентов в списке.

Строка 286 – условие «поиска» цикла строк 285 – 286. Если в одной из ячеек массива по openLS имеет место значение, тождественное значению, «выданному» элементом «Список студентов» при нажатии кнопки «Выбор» - выход из цикла, переход на ранее рассмотренную метку end . Результат выполнения операторов строк 285, 286 - блокировка повторного формирования билета для одного и того же студента.

Строка 288 - 289 – формирование первой строки массива tt[0] (определён в строке 89) - первой строки экзаменационного билета (Билет № - Ф. И. О.).

Строки 291 - 313 – программная конструкция switch . Аргумент switch переменная nq - число вопросов в билете. Назначение – формирование экзаменационного билета.

Аналогично пройденному в главе 3 рассмотрим два первых варианта case из четырёх:

case 1: вариант при задании одного вопроса в билете. В теле case 1 имеют место три оператора:

Строка 293 – 1-ый оператор блока – вызов функции choice(q1, n1, 0) с передачей ей трёх аргументов: q1 - хендлер на массив вопросов 1-го блока (выбор первого вопроса); n1 - количествовопросов в первом блоке; индекс столбца массива по openQ (определён в строке 99, инициирован в строке 265).

Очевидно, что для рассмотрения дальнейшей работы программы необходим переход к определению функции choice() , такой переход Вам и рекомендуется проделать (глава 6, страница 18).

После завершения работы каждого из заданных вызовов функции choice() в любом из четырёх возможных case выполняется оператор «занесения» значения в ячейку массива по хендлеру tt (определён в строке 89, инициирован в строке 278). В ячейки по индексам 1 – 4 «заносится» конкатенант: номер вопроса – экзаменационный вопрос соответствующего блока. Далее break - переход на строку 314.

Строка 315 – вывод массива элементов lbl[k-1] (label) в количестве, соответствующем количеству студентов с установкой одного из стилей.

Строка 316 – вызов функции fw() - запись в файл сформированного экзаменационного билета в файл. В этой функции, кроме этого, осуществляется манипуляция переменной-счётчиком k (определён в строке 96) с его инкрементом.

Должно быть очевидно понятным, что с этого места учебного пособия нужно «переключиться» на изучение организации работы функции fw() (глава 7, страница 20).

Строка 317 – обновить изображение.

6. Функция choice() – рандомизированного выбора вопроса

Строка 360 – заголовок определения функции choice() . В списке аргументов, в стиле исходного C определены три локальные переменные функции: ars , ns , ia .

Строки 361 - 372 – тело определения функции.

Строка 362 – определение хендлера r на класс System::Random генерации случайных чисел; определение переменной ri .

Строка 364 – «по-простому» /6/ - инициализация генерации случайных чисел, «по-сложному» - задание garbage collected heap для манипуляции рандомизируемыми числами; в строке установлена метка rr .

Строка 365 – через хендлер r вызывается метод Next() объекта класса Random с передачей ему в качестве аргумента значения, присвоенного локальной переменной ns функции choice при её вызове (при вызовах функции в качестве этого аргумента передаётся одно из значений n1, n2, n3, n4 – число вопросов в конретном блоке

вопросов). Результат выполнения оператора данной строки – присвоение переменной ri любого значения из диапазона числа вопросов в блоке экзаменационных вопросов.

Строки 367 - 368 – цикл для «выявления» повторного «выпадания» вопроса.

Строка 368 – если вопрос «выпал» повторно – переход на строку 364 по метке rr с повторением действий по выбору номера вопроса. Массив openQ[] как раз и предназначен для заполнения матрицы выбранных вопросов.

Строка 370 – в ячейку объекта класса String по хендлеру st (определён в строке 90) записывается строковый литерал вопроса. Строка эта считывается из ячейки массива с индексом ri по локальному хендлеру ars (в этот массив при вызове функции choice() передаётся содержимое одного из массивов блоков вопросов q1, q2, q3, q4).

Строка 371 – в ячейку двухмерного массива int по хендлеру openQ (определён в строке 99) записывается значение переменной ri (сгенерированный случайным образом номер вопроса). При этом, индекс строки k-1 - индекс студента/билета, выбранного в порядке очерёдности (переменная k , определённая в строке 96, содержит порядковый номер – студент - №Билета (инкремент порядкового номера «студент – билет» строка 403); индекс столбца - переменная ia - третий аргумент, переданный функции choice() при вызове – её значением «жёстко» задаётся индекс столбца двухмерного массива openQ (индекс «студент – билет» – номера вопросов по блокам).

Например, рассмотрим первый вызов функции choice(q1, n1, 0) (глава 5, строка 293): локальному хендлеру ars передаётся «нацеленность» на 1-ый блок вопросов; локальной переменной ns передаётся число вопросов в 1-ом блоке; локальной переменной ia передаётся индекс столбца двухмерного массива openQ (индекс столбца 0 –один вопрос в билете). Т.е., при первом таком вызове функции choice() в ячейку openQ[0, 0] (первый выбор билета) будет занесено любое число из заданного диапазона числа вопросов первого блока (это число - индекс вопроса в блоке вопросов).

Для возможного большего понимания механизма действия изучаемой программной конструкции рассмотрим второй пример. Пусть в билете назначено два вопроса; в 1-ом блоке вопросов – 21 вопрос; во 2-ом – 22 вопроса; студент – седьмым выбрал билет. Т.е.,

переменная счётчик k имеет значение 7 - инкремент счётчика выбранных билетов задан в функции fw() (глава 7, страница 20; функция вызывается при выборе билета – глава 5, страница 16). Этому примеру соответствует вызов функции choice() по case 2 (глава 5, строка 295). Причём, в соответствии с заданным числом вопросов, в этом случае функция choice() вызывается дважды с разными аргументами (choice(q1, n1, 0) , choice(q2, n2, 1)). В результате в ячейку openQ[6, 0] может быть записано значение - 20, а в ячейку openQ[6, 1] значение – 5.

7. Функция fw()– записи билетов в файл

Строки 377 - 405 – тело определения функции.

Строка 379, 380, 381 – определение локализованного в рассматриваемой функции хендлера с идентификатором fi на объект класса «системный ввод/вывод – информация о файле» с его инициализацией. Дальнейшие синтаксические конструкции должны быть уже легко читаемыми. Имя файла и его дислокация определены «жёстко».

Строка 383 – определение локального хендлера с идентификатором sw на объект класса «системный ввод/вывод – поток записи в файл».

Строки 385 - 390 – блок оператора if… else . Если файл для записи имеет место fi->Exists , оператором строки 387 открывается поток для записи в файл.

Строка 390 – если назначенного файла нет – он создаётся и для него открывается поток записи.

Строки 394 - 398 – цикл построчной записи в файл. В аргументах for реализовано: итерация индекса строки от 0 (int i = 0) до количества вопросов в блоке (i < (nq+1)) с единичным инкрементом (i++). В теле цикла происходит следующее:

Строка 395 – через поток sw с вызовом его метода WriteLine() в файл, «сопоставленный» потоку ранее заносится строка массива tt[i] , соответствующая шагу итерации цикла.

Строка 396 – вывести в элемент listBox1 через вызов метода Add() его свойства Items содержимого соответствующей строки массива tt[i] .

Строка 397 – ячейке массива tt[i] присваивается «пустая строка».

В результате выполнения цикла в файл записывается содержимое персонифицированного экзаменационного билета, а в элементе listBox1 оно отображается. После реализации записи и вывода сформированного билета выполняется последовательность дополнительных операций.

Строка 399 – запись «пустой строки» в файл – визуальный разграничитель билетов в тестовом документе.

Строка 400 – добавление «пустой строки» в элемент listBox1 – визуальный разграничитель билетов в тексте этого элемента.

Строка 402 – в ячейку массива openLS[k-1] (индекс ячейки - k-1 соответствует «порядковому номеру студент – билет» - k) записывается индекс студента по списку студентов, студента в любой очерёдности выбравшего билет. Значение индекса студента, «сформировавшего» билет «отдаётся» свойством comboBox1->SelectedIndex .

Строка 403 – инкремент переменной-счётчика k нумерации студентов (строка 96) студентов в порядке выбора билета (первым взял – получил 1-ый номер и билет №1).

Строка 404 – закрытие потока для записи в файл.

8. Функция drawPI() – использование графической плоскости

Напоминаем о том, что рассматриваемая программная разработка создавалась в учебных целях. Но, в то же время, она имеет определённый практический смысл. Пример «по работе с графикой» в VC++ CLR первоначально задумывался как учебная демонстрация. Но, если попытаться вдуматься в его функционал, можно, при удачном стечении обстоятельств, придти к мыслям о том, что в примере реализован элемент инженерного (технического) дизайна. Т.е., дизайна ни в коей мере не оформительского (геометрия, цвета, стили и т.д. в примере, с этой точки зрения, не оптимизированы). В чём заключается смысловое содержание технического (инженерного) дизайна рассматриваемого примера использования графической поверхности. Ответ на этот вопрос – функционал этого элемента программы: «графический» вывод массивов (количество массивов - количество блоков вопросов) прямоугольников (количество прямоугольников - количество вопросов в блоке); выбор билета визуализируется выделением соответствующих прямоугольников. С точки зрения технической эргономики можно рассматривать эту программную реализацию (программный эскиз) как «заготовку» информационной панели о ходе выбора билетов. С точки зрения технической же эргономики, в концепциях визуализации для пользователя работы программного обеспечения, данный «программный эскиз» в какой-то степени «параллелен» стандартному элементу ProgressBar , смысловое содержание функционала которого заключается в визуализации хода выполнения процесса и, самое главное, в отвлечении внимания пользователя.

Рассмотрим «программный эскиз», реализованный функцией drawPI() .

Строка 407 – заголовок определения функции drawPI() . Аргументы аналогичны аргументам, рассмотренным в 3-ей главе (страница 12). Только имя класса EventArgs^ e более конкретизировано наследником PaintEventArgs .

Строки 408 - 457 – тело определения функции drawPI() .

Строка 410 – определение графической поверхности с «идентификатором программиста» gq /5/. С точки зрения нашего анализа «обновлённого синтаксиса C++» (базируется то он на

«классическом»), в этой строке определён хендлер на «построение графики» - объекта, наследующего в иерархии системные классы (System::Drawing::Graphics^ gq). Далее этому хендлеру присваивается значение системного хендлера (входного аргумента Forms::PaintEventArgs^ e), который обращается к методу «графика».

Строка 411 – определение переменной yg для «текущей» координаты по оси ординат.

Строка 412 – определение переменной xg0 для начального значения по оси абсцисс. Инициализируется при определении 16-ти разрядным целым. В аргументе преобразования к этому формату имеет место обращение к членам класса формы через указатель this . Конкретнее, используется «клиентская ширина» окна с уменьшающим коэффициентом.

Строка 413 – определение переменной yg0 для начального значения по оси ординат. Смысловое содержание инициализатора переменной, по смысловому содержанию аналогично предыдущей строке.

Строка 414 – определение переменной xg для «текущей» координаты по оси ординат абсцисс. При определении инициализируется начальной координатой.

Строка 416 – определение и инициализация переменной wg для ширины графического элемента (прямоугольник – индикация одного вопроса).

Строка 417 – определение и инициализация переменной hg для определения высоты вышеобозначенного «графического индикатора».

Строка 419 – «внешний» цикл вывода «графических индикаторов» на окно (форму). Тело цикла – строки 420 – 437. Условия итерации: постинкремент счётчика цикла от 0 до индекса, определяемого количеством вопросов в билете (например, задано 2 вопроса в билете: конечный индекс цикла – 1; число итераций – 2). Этот «наружный» цикл формирует столбцы «графических индикаторов».

Строка 421 – переменной yg («текущая» координата по оси y) присваивается начальное значение, т.е., каждый столбец графических элементов начинает формироваться с начального значения yg0.

Строка 422 – «внутренний» (вложенный) цикл вывода «графических индикаторов» на окно (форму). Цикл формирует

«строки» «графической матрицы», отображающей процесс формирования билетов.

Тело цикла – строки 423 – 435. Условия итерации: постинкремент счётчика цикла от 0 до индекса, определяемого значением наибольшего количества вопросов в одном из блоков вопросов - nn (она определена в строке 95). Т.е., конечный индекс цикла nn - 1.

Строки 424 - 427 – «последовательность» операторов «принятия решения» - если число вопросов в блоке (точнее, индекс блока вопросов) становится меньше значения счетчика цикла «построения строк» и «это подтверждено значением индекса внешнего цикла (индексом столбца, т.е., «какой столбец выводится»), оператор break «досрочно» заканчивает цикл – переход на строку 436. Результат – для каждого блока вопросов будет выведено ровно столько графических элементов, сколько вопросов в этом блоке. Эта последовательность операторов, по прежнему «жёстко» ориентирована на максимальное количество вопросов в билете – 4.

Строка 428 – элемент numericUpDown1 «блокируется» после выбора первого билета (строка 280). После этого изменение числа вопросов в билете становится невозможным. Если numericUpDown1 «заблокирован» оператор безусловного перехода goto переводит алгоритм на строку 432 по метке lps . Тем самым исключается выполнение последовательности операторов строк 429 – 431, поскольку массив «координат» графических элементов openQG уже заполнен.

Строки 429 - 431 – последовательность операторов записи «начальных координат» графических элементов в массив openQG .

Строка 429 – в ячейку «первой таблицы массива» openQG с индексом строки j и индексом столбца i записывается значение индекса строки - j .

Строка 430 – в ячейку «второй таблицы массива» openQG с индексом строки j и индексом столбца i записывается значение «текущей» координаты начала вывода графического элемента по оси абсцисс - xg.

Строка 431 – в ячейку «третьей таблицы массива» openQG с индексом строки j и индексом столбца i записывается значение «текущей» координаты начала вывода графического элемента по оси ординат- yg.

Как индексно идентифицируются графические элементы с соответствующими координатами (размер «графической матрицы»: число строк - максимальное число вопросов; максимальное число столбцов – 4). В «первую таблицу» массива в ячейки с индексами строк j и индексами столбцов i записываются индексы строк j соответсвующего столбца матрицы. Число строк в каждом столбце матрицы индивидуально (количество вопросов в блоке), поэтому индекс строки j каждого элемента и подлежит сохранению. Индекс столбца i графического элемента соответствует индексу столбца массива openQG , поэтому в выделении ячеек памяти не нуждается. Именно поэтому массив openQG задан 3-ёх мерным, а не 4-ёх мерным.

Строка 433 – через хендлер gq вызывается метод DrawRectangle соответствующего базового класса. Полученному в результате объекту присваиваются свойства, имеющие место в списке параметров вызова функции – метода (в примере: «жёлтое перо», описанная ранее геометрия).

Строка 434 – задаётся смещение каждого следующего прямоугольника в столбце «вниз» на пять пикселей от «нижней» границы предыдущего элемента.

Строка 436 – перед выполнением итерации «цикла по столбцам» задаётся смещение координаты вывода следующего столбца на пять пикселей «вправо» от «правой границы» предыдущего столбца.

Строка 439 – Если не выбран первый билет элемент numericUpDown1 «не заблокирован» - возможность изменения числа вопросов в билете «открыта». Начальное число numericUpDown1 - 1. Т.е., функцией drawPI() при запуске программы будет выведен один столбец графических элементов в количестве, соответствующем количеству вопросов в первом блоке. Поскольку формирования 1-го билета не произошло, выполнение операторов, задающих выделение графических элементов, соответствующих назначенным вопросам, «блокируется» - безусловный переход goto на метку lps1 (строка 455).

Строки 441 - 454 – последовательность операторов, определяющих выделение прямоугольников, отображающих выбранные вопросы.

Строка 441 – внешний цикл «перебора» графических элементов (и «массивов контроля») по столбцам. Тело цикла – строки 442 – 454.

Цикл аналогичен циклу, заданному строкой 419. Счётчик цикла i индексирует столбец массива openQ[j, i] в условном операторе строки 448 и столбцы массива openQG .

Строка 443 – цикл в цикле «первого уровня вложения» для «перебора» индексов студентов группы. Тело цикла – строки 444 – 453. Условия итерации: постинкремент счётчика цикла от 0 до индекса, определяемого значением stu -число студентов в списке. Счётчик цикла j индексирует строку массива openQ[j, i] в условном операторе строки 448.

Строка 445 – цикл в цикле «второго уровня вложения» для «перебора» индексов графических элементов по строкам. Тело цикла – строки 446 – 452. Цикл аналогичен циклу, заданному строкой 422. Счётчик цикла m индексирует строки массива openQG .

Строка 448 – в условии оператора if имеет место компарирование содержимого ячеек двух массивов: двухмерного openQ[j, i] и трёхмерного openQG[m, i, 0] . Матрица openQ[j, i] (определена в строке 99) содержит в своих ячейках индексы вопросов. Индекс строки – персонифицированный выбор билета, индекс столбца – индекс блока вопросов. Содержимое матрицы openQG рассмотрено в этой главе с использованием описания её заполнения - строки 429 – 431.

Кратко, смысл этого «трёхмерного сканирования» массивов со сравнением значений их ячеек заключается в «обнаружении» вопросов экзаменационных блоков, поставленных в соответствие персонифицированному билету. В случае «нахождения» такого объекта сканирования, выполняется графическое выделение соответствующего графического объекта методом FillRectangle с вызовом в качестве аргументов и требуемых координат графического элемента openQG[m, i, 1], openQG[m, i, 2], wg, hg .

Строка 455 – метка lps1 фактически – выход из функции.

Строка 456 – «пустой оператор», без последующего оператора метка «не работоспособна» - наследие «классического» C/C++.

Заключение

Рассмотренное приложение создано разработчиком «эскизно» с целями начала освоения предметной области и в качестве учебного примера (на оптимальность программные решения не претендуют). Доработка данного приложения до полностью «профессионального уровня» требует на порядки большего ресурсопотребления. В этой связи, конечно же, рассмотрены далеко не все аспекты технологии программирования в VC++ CLR.

Список использованных источников

1. Св. РФ о гос. рег. пр. ЭВМ №2013611613. Формирование экзаменационных билетов. / Д. Д. Ветчинин. ИГТА. - Зарег. реестр. пр. ЭВМ 29.01.13.

2. Д. Д. Ветчинин. Введение в VC++ MFC: Учебное пособие. – Иваново, ИГТА, 2012г.– 104 с.

3. Д. Д. Ветчинин. Основы C++ - классы: Учебное пособие. – Иваново, ИГТА, 2013г.– 56 с.

4. Х. М. Дейтел, П. Дж. Дейтел, Т. Р. Нието. Visual Basic.NET: Книга 1, Основы программирования. – М.: «Издательство БИНОМ», 2003. - 765 с.:ил.

5. Документация Microsoft Visual Studio 2008.

6. Н. Б. Культин. Microsoft Visual C++ в задачах и примерах. – СПб.: БХВ-Петербург, 2010г. – 272 с.:ил.

Приложение 1

```
1   #pragma once
2
3   namespace et {
4
5   using namespace System;
6   using namespace System::ComponentModel;
7   using namespace System::Collections;
8   using namespace System::Windows::Forms;
9   using namespace System::Data;
10  using namespace System::Drawing;
11
12  /// <summary>
13  /// Сводка для Form1
14  /// Внимание! При изменении имени этого класса необходимо также изменить
15  /// свойство имени файла ресурсов ("Resource File Name") для средства
16  /// компиляции управляемого ресурса,связанного со всеми файлами с
17  /// расширением .resx, от которых зависит данный класс.
18  /// В противном случае,конструкторы не смогут правильно работать с
19  /// локализованными ресурсами, сопоставленными данной форме.
20  /// </summary>
21
22  public ref class Form1 : public System::Windows::Forms::Form
23  {
24     public:
25        Form1(void)
26        {
27            InitializeComponent();
28              System::IO::StreamReader^fr; // поток для чтения
29
30              try {           // выброс исключения (любая ошибка)
31               // создание потока для чтения из файла (список студентов)
32               fr = gcnew System::IO::StreamReader(
33                  Application::StartupPath + "\\Stu.txt",
34                        System::Text::Encoding::GetEncoding(1251));
35              }               // конец блока try
36              catch (Exception^ )         {
37                      MessageBox::Show("Файла 'Stu.txt' НЕТ!\n", "Электронный билет",
38                      MessageBoxButtons::OK, MessageBoxIcon::Error);
39                      return;
40              }               // конец блока catch
41
42               // чтение файловых данных
43               int i = 0;
44               stu = Convert::ToInt16(fr->ReadLine());
45               ls = gcnew array<String^>(stu);
```

```
46              openLS = gcnew array<int>(stu);
47
48              int x1 = numericUpDown1->Bottom;
49              int y1 =5, lw = 25, lh = 25, dx = 2;
50
51              lbl = gcnew array<System::Windows::Forms::Label^>(stu);
52
53            try {           //***2012
54              while(!fr->EndOfStream)
55              {
56                 lbl[i] = gcnew System::Windows::Forms::Label;
57                 lbl[i]->SetBounds(x1, y1, lw, lh);
58                 lbl[i]->BackColor = System::Drawing::Color::GreenYellow;
59                 lbl[i]->Text = (i+1).ToString();
60                 this->Controls->Add(lbl[i]);
61                 x1 = x1 + lw + dx;
62
63                 ls[i] = fr->ReadLine();
64                 comboBox1->Items->Add(ls[i++]);
65              }               //***2012
66            }
67            catch (Exception^ )         {               //***2012
68                  MessageBox::Show("Файл 'Stu.txt' сформирован с ошибками!\n",
69                  "Электронный билет", MessageBoxButtons::OK,
70                  MessageBoxIcon::Error);
71                  return;
72            }               //***2012
73
74              fr->Close(); // закрытие потока
75              comboBox1->SelectedIndex = 0;
76
77              // элемент выбора числа вопросов
78              numericUpDown1->Maximum = 4;
79              numericUpDown1->Minimum = 1;
80              numericUpDown1->TabStop = false;
81
82              // графическая плоскость (поле?)
83              this->Paint += gcnew
84                 System::Windows::Forms::PaintEventHandler(this, &Form1::drawPI);
85       }
86
87  private:
88     static int nq; static int stu;
89     array<String^>^ ls; array<String^>^ tt;
90     array<String^>^ q;  int n;  String^ st;
91
92     array<String^>^ q1;    array<String^>^ q2;
93     array<String^>^ q3;    array<String^>^ q4;
94
95     int n1, n2, n3, n4, nn;
```

```
96    int static k = 1;
97
98    static array<int>^ openLS;
99    static array<int,2>^ openQ;
100   static array<int,3>^ openQG;
101
102   array<System::Windows::Forms::Label^> ^lbl;
103
104 //System::Windows::Forms::Button^  button1;
105 System::Windows::Forms::ComboBox^  comboBox1;
106 System::Windows::Forms::NumericUpDown^  numericUpDown1;
107 private: System::Windows::Forms::Button^  button1;
108 private: System::Windows::Forms::ListBox^  listBox1;
109                 System::ComponentModel::Container ^components;
110
111
112 #pragma region Windows Form Designer generated code
113 /// <summary>
114 /// Обязательный метод для поддержки конструктора - не изменяйте
115 /// содержимое данного метода при помощи редактора кода.
116 /// </summary>
117
118   void InitializeComponent(void)
119   {
120           System::ComponentModel::ComponentResourceManager^  resources =
121             (gcnew System::ComponentModel::
122             ComponentResourceManager(Form1::typeid));
123           this->comboBox1 = (gcnew System::Windows::Forms::ComboBox());
124           this->numericUpDown1 = (gcnew
125             System::Windows::Forms::NumericUpDown());
126           this->button1 = (gcnew System::Windows::Forms::Button());
127           this->listBox1 = (gcnew System::Windows::Forms::ListBox());
128             (cli::safe_cast<System::ComponentModel::ISupportInitialize^->;
129             (this->numericUpDown1))->BeginInit()
130           this->SuspendLayout();
131           //
132           // comboBox1
133           //
134           this->comboBox1->BackColor = (static_cast<System::Byte>(192)),
135             System::Drawing::Color::FromArgb(static_cast<System::Int32>
136             static_cast<System::Int32>(static_cast<System::Byte>(255)),
137             static_cast<System::Int32>(static_cast<System::Byte>(192)));
138           this->comboBox1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
139             Serif", 13.8F, System::Drawing::FontStyle::Bold,
140             System::Drawing::GraphicsUnit::Point, static_cast<System::Byte>(204)));
141           this->comboBox1->FormattingEnabled = true;
142           this->comboBox1->Location = System::Drawing::Point(75, 221);
143           this->comboBox1->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);
144           this->comboBox1->Name = L"comboBox1";
145           this->comboBox1->Size = System::Drawing::Size(507, 37);
```

```
146             this->comboBox1->TabIndex = 1;
147             //
148             // numericUpDown1
149             //
150             this->numericUpDown1->BackColor = System::Drawing
151                 ::Color::FromArgb(static_cast<System::Int32>
152                 (static_cast<System::Byte>(192)),
153                 static_cast<System::Int32>(static_cast<System::Byte>(255)),
154                 static_cast<System::Int32>(static_cast<System::Byte>(192)));
155             this->numericUpDown1->BorderStyle =
156                 System::Windows::Forms::BorderStyle::FixedSingle;
157             this->numericUpDown1->Font = (gcnew System::Drawing::Font(L"Microsoft
158                 Sans Serif", 13.8F, System::Drawing::FontStyle::Bold,
159                 System::Drawing::GraphicsUnit::Point, static_cast<System::Byte>(204)));
160             this->numericUpDown1->Location = System::Drawing::Point(75, 112);
161             this->numericUpDown1->Margin = System::Windows::Forms::
162                 Padding(3, 4, 3, 4);
163             this->numericUpDown1->Name = L"numericUpDown1";
164             this->numericUpDown1->Size = System::Drawing::Size(63, 34);
165             this->numericUpDown1->TabIndex = 2;
166             this->numericUpDown1->ValueChanged += gcnew
167                 System::EventHandler(this, &Form1::numericUpDown1_ValueChanged);
168             //
169             // button1
170             //
171             this->button1->FlatStyle = System::Windows::Forms::FlatStyle::Flat;
172             this->button1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
173                 Serif", 13.8F, System::Drawing::FontStyle::Bold,
174                 System::Drawing::GraphicsUnit::Point,
175                 static_cast<System::Byte>(204)));
176             this->button1->Location = System::Drawing::Point(265, 112);
177             this->button1->Name = L"button1";
178             this->button1->Size = System::Drawing::Size(317, 52);
179             this->button1->TabIndex = 3;
180             this->button1->Text = L"ВЫБОР";
181             this->button1->UseVisualStyleBackColor = true;
182             this->button1->Click += gcnew System::EventHandler(this,
183                 &Form1::button1_Click);
184             //
185             // listBox1
186             //
187             this->listBox1->BackColor =
188                 System::Drawing::Color::FromArgb(static_cast<System::Int32>
189                 (static_cast<System::Byte>(192)),
190                 static_cast<System::Int32>(static_cast<System::Byte>(255)),
191                 static_cast<System::Int32>(static_cast<System::Byte>(192)));
192             this->listBox1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
193                 Serif", 12, System::Drawing::FontStyle::Bold,
194                 System::Drawing::GraphicsUnit::Point, static_cast<System::Byte>(204)));
195             this->listBox1->FormattingEnabled = true;
```

```
196             this->listBox1->HorizontalExtent = 4500;
197             this->listBox1->HorizontalScrollbar = true;
198             this->listBox1->ItemHeight = 25;
199             this->listBox1->Location = System::Drawing::Point(75, 365);
200             this->listBox1->Name = L"listBox1";
201             this->listBox1->Size = System::Drawing::Size(918, 379);
202             this->listBox1->TabIndex = 4;
203             //
204             // Form1
205             //
206             this->AutoScaleDimensions = System::Drawing::SizeF(9, 18);
207             this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
208             this->BackColor =
209                 System::Drawing::Color::FromArgb(static_cast<System::Int32>
210                 (static_cast<System::Byte>(192)),
211                 static_cast<System::Int32>(static_cast<System::Byte>(255)),
212                 static_cast<System::Int32>(static_cast<System::Byte>(192)));
213             this->BackgroundImage = (cli::safe_cast<System::Drawing::Image^ ->
214                 (resources->GetObject(L"$this.BackgroundImage")));
215             this->BackgroundImageLayout =
216                 System::Windows::Forms::ImageLayout::Zoom;
217             this->ClientSize = System::Drawing::Size(1027, 773);
218             this->Controls->Add(this->listBox1);
219             this->Controls->Add(this->button1);
220             this->Controls->Add(this->numericUpDown1);
221             this->Controls->Add(this->comboBox1);
222             this->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9,
223                 System::Drawing::FontStyle::Regular,
224                 System::Drawing::GraphicsUnit::Point, static_cast<System::Byte>(204)));
225             this->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);
226             this->Name = L"Form1";
227             this->WindowState =
228                 System::Windows::Forms::FormWindowState::Maximized;
229             (cli::safe_cast<System::ComponentModel::ISupportInitialize^ ->
230                 (this->numericUpDown1))->EndInit();
231             this->ResumeLayout(false);
232
233     }
234
235 #pragma endregion
236
237 private: System::Void numericUpDown1_ValueChanged(System::Object^ sender,
238                                 System::EventArgs^ e)
239     {
240         nq = Convert::ToInt16(numericUpDown1->Value);
241         n1 = n2 = n3 = n4 = 0;
242       switch (nq)
243             {
244                     case 1:
245                                 { fq("\\q1.txt");   n1 = n; q1 = q; } break;
```

```
246                     case 2:
247                             { fq("\\q1.txt");   n1 = n; q1 = q;
248                                     fq("\\q2.txt"); n2 = n; q2 = q; } break;
249                     case 3:
250                             { fq("\\q1.txt");   n1 = n; q1 = q;
251                                     fq("\\q2.txt"); n2 = n; q2 = q;
252                                     fq("\\q3.txt"); n3 = n; q3 = q; } break;
253                     case 4:
254                             { fq("\\q1.txt");   n1 = n; q1 = q;
255                                     fq("\\q2.txt"); n2 = n; q2 = q;
256                                     fq("\\q3.txt"); n3 = n; q3 = q;
257                                     fq("\\q4.txt"); n4 = n; q4 = q; } break;
258             }
259
260             nn = n1;
261             if(n2 > nn) nn = n2;
262             if(n3 > nn) nn = n3;
263             if(n4 > nn) nn = n4;
264
265             openQ = gcnew array<int,2>(stu,nq);
266             openQG = gcnew array<int,3>(nn,nq,3);
267
268             for(int l = 0; l < nq; l++) {
269                     for(int i = 0; i < stu; i++) {openQ[i,l] = nn + 1;} }
270
271             this->Refresh();
272
273     }
274
275 private: System::Void button1_Click(System::Object^ sender,
276                     System::EventArgs^ e)
277             {
278               tt = gcnew array<String^>(nq + 1);
279
280               if(k == 1) { numericUpDown1->Enabled = false;
281                     for(int i = 0; i < stu; i++) openLS[i] = stu;}
282
283               if(k > stu) goto end;
284
285               for(int i = 0; i < stu; i++)              {
286                     if(openLS[i] == comboBox1->SelectedIndex) goto end;         }
287
288                     tt[0] = "Билет " + k.ToString() + " - "
289                             + comboBox1->Text;
290
291                     switch (nq)
292                     {
293                     case 1: {choice(q1, n1, 0);
294                             tt[1] = "1. " + st; break;}
295                     case 2: {choice(q1, n1, 0);
```

```
296                             tt[1] = "1. " + st;
297                             choice(q2, n2, 1);
298                             tt[2] = "2. " + st; break;}
299                     case 3: {choice(q1, n1, 0);
300                             tt[1] = "1. " + st;
301                             choice(q2, n2, 1);
302                             tt[2] = "2. " + st;
303                             choice(q3, n3, 2);
304                             tt[3] = "3. " + st; break;}
305                     case 4: {choice(q1, n1, 0);
306                             tt[1] = "1. " + st;
307                             choice(q2, n2, 1);
308                             tt[2] = "2. " + st;
309                             choice(q3, n3, 2);
310                             tt[3] = "3. " + st;
311                             choice(q4, n4, 3);
312                             tt[4] = "4. " + st; break;}
313                     }
314
315                     lbl[k-1]->BackColor = System::Drawing::Color::Azure;
316                     fw();
317                     this->Refresh();
318 end:
319              ;
320             }
321
322 // функция считывания вопроса из файла
323 void fq(String^ f)
324 {
325             System::IO::StreamReader^fr; // определение потока для чтения
326   try {             //***2012
327             // создание потока для чтения из файла (один из файлов вопросов)
328             fr = gcnew System::IO::StreamReader(
329                     Application::StartupPath+f,
330                     System::Text::Encoding::GetEncoding(1251));
331
332                     n = Convert::ToInt16(fr->ReadLine());
333                     int i = 0;
334                     q = gcnew array<String^>(n);
335
336             while(!fr->EndOfStream)
337             {
338                     q[i++] = fr->ReadLine();
339             }
340   }                         //***2012
341   catch (System::IO::FileNotFoundException^ ) {         //***2012
342               MessageBox::Show("НЕТ файла данных! - \n" +
343               Application::StartupPath + f, "Электронный билет",
344               MessageBoxButtons::OK, MessageBoxIcon::Error);
345               numericUpDown1->Value = 1;
```

```
346             return;
347       }                         //***2012
348
349       catch (System::Exception^ )       {         //***2012
350               MessageBox::Show("Ошибка в файле данных! - \n" +
351               Application::StartupPath + f, "Электронный билет",
352               MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
353               numericUpDown1->Value = 1;
354             return;
355       }                         //***2012
356             fr->Close(); // закрытие потока
357   }
358
359   // функция рандомизированного выбора (студент – вопрос)
360   void choice(array<String^>^ ars, int ns, int ia)
361   {
362             Random^ r; int ri;
363
364   rr:       r = gcnew Random();
365                       ri = r->Next(ns);
366
367             for(int i = 0; i < stu; i++) {
368                       if(openQ[i, ia] == ri) goto rr;}
369
370             st = ars[ri];
371             openQ[k-1, ia] = ri;
372   }
373
374
375   //  функция записи сформированного билета в файл
376   void fw()
377   {
378              //получить информацию о файле
379              System::IO::FileInfo^ fi =
380                       gcnew System::IO::FileInfo(
381                       Application::StartupPath + "\\tickets.txt");
382              //поток для записи
383              System::IO::StreamWriter^ sw;
384
385              if (fi->Exists)
386                       //открыть поток для добавления в существующий файл
387                       sw = fi->AppendText();
388              else
389                       //создать файл и открыть поток для записи
390                       sw = fi->CreateText();
391
392              //запись в файл
393              for(int i = 0; i < (nq+1); i++)
394              {
395                       sw->WriteLine(tt[i]);
```

```
396                     listBox1->Items->Add(tt[i]);
397                     tt[i] = "";
398             }
399                     sw->WriteLine("");
400                     listBox1->Items->Add("");
401
402             openLS[k-1] = comboBox1->SelectedIndex; //уже выбран
403             ++k;
404             sw->Close();
405     }
406
407     void drawPI(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e)
408     {
409             //графическая поверхность
410             System::Drawing::Graphics^ gq = e->Graphics;
411             int yg;
412             int xg0 = Convert::ToInt16((this->ClientSize.Width) * 0.9);
413             int yg0 = Convert::ToInt16((this->ClientSize.Height) * 0.01);
414             int xg = xg0;
415
416             int wg = Convert::ToInt16((this->ClientSize.Width) * 0.015);
417             int hg = Convert::ToInt16((this->ClientSize.Height) * 0.02);
418
419             for(int i = 0; i < nq; i++)
420             {
421                     yg = yg0;
422                     for(int j = 0; j < nn; j++)
423                     {
424                             if(((n1-1) < j) && (i == 0)) break;
425                             if(((n2-1) < j) && (i == 1)) break;
426                             if(((n3-1) < j) && (i == 2)) break;
427                             if(((n4-1) < j) && (i == 3)) break;
428             if(numericUpDown1->Enabled == false) goto lps;
429                                     openQG[j, i, 0] = j;
430                                     openQG[j, i, 1] = xg;
431                                     openQG[j, i, 2] = yg;
432     lps:
433                             gq->DrawRectangle(Pens::Yellow, xg, yg, wg, hg);
434                             yg += (hg + 5);
435                     }
436                     xg += (wg + 5);
437             }
438
439             if(numericUpDown1->Enabled == true) goto lps1; //***2012
440
441             for(int i = 0; i < nq; i++)
442             {
443                     for(int j = 0; j < stu; j++)
444                     {
445                             for(int m = 0; m < nn; m++)
```

```
446                             {
447
448                                 if(openQ[j, i] == openQG[m, i, 0]) gq->FillRectangle
449                                     (System::Drawing::Brushes::Magenta,
450                                     openQG[m, i, 1], //openQG[j, i, 1] //***2012
451                                     openQG[m, i, 2], wg, hg);
452                             }
453                         }
454                     }
455 lps1:               //***2012
456  ;                          //***2012
457 }
458
459 };
460 }
```

Содержание

№ п/п	Наименование	Стр.
	Введение	3
1.	Этап визуального проектирования	4
2.	Конструктор формы Form1	5
3.	Функция – обработчик события «задание числа вопросов в билете»	12
4.	Функция fq() – считывания вопросов из файлов	15
5.	Функция – обработчик события «выбрать билет»	16
6.	Функция choice() – рандомизированного выбора вопроса	18
7.	Функция fw()– записи билетов в файл	20
8.	Функция drawPI() – использование графической плоскости	22
	Заключение	27
	Список использованных источников	28
	Приложение 1	29
	Содержание	39

Учебное пособие по курсам «Прикладное программирование в информационных системах», «Программная реализация средств дизайна», «Технологии программирования»
Основы работы в среде VC++ CLR
Составители: Дмитрий Давидович Ветчинин
Научный редактор: Николай Анатольевич Коробов
Печатается в авторской редакции

УП № 032347 от 23.09.2013. Подписано в печать 24.09.2013.
Формат 1/16 60*84. Бумага писчая. Плоская печать.
Усл. печ. л. 2,12 Уч.-изд.7,062. Тираж 50 экз. Заказ № 537

РИО ИвТИ
ЦОТ кафедры ВПМСИТ УЦСГЕН
152000, г. Иваново пр. Шереметьевский 21