УДК 621.396.2

К.Н. Яковишин

Национальный авиационный университет, Киев

УСКОРЕННОЕ ОБУЧЕНИЕ ПРОГРАММИРОВАНИЮ В СРЕДЕ BORLAND C++BUILDER ПРОФЕССИОНАЛОВ ДЛЯ ТЕЛЕКОММУНИКАЦИЙ

Профессионал по телекоммуникациям должен знать и уметь моделировать телекоммуникационные системы и протоколы в средах быстрого проектирования методами объектно-ориентированного программирования. В связи с этим рассматривается техника ускоренного обучения программированию на языке C++ в среде Borland C++Builder. В основе этой техники лежат следующие принципы. Обучение начинается с примера создания работающего приложения путем «мавпування» (подражания) действий, описанных в самоучителе. Используется идея восприятия студентом знаний и формирования навыков на подсознательном уровне (по аналогии с техникой обучения языкам в режиме 25-го кадра). Затем обучение продолжается проведением все более и более детального анализа структуры проекта (приложения). Наконец, на этапе контроля студенты успешно создают новое собственное приложение по аналогии с изученным примером. В результате за полтора-два месяца студенты с нулевого уровня знаний и навыков достигают уровня самостоятельного создания приложений.

***Ключевые слова**: техника ускоренного обучения программированию, восприятие знаний и формирование навыков на подсознательном уровне, техника обучения языкам в режиме 25-го кадра.*

Введение

Постановка проблемы. По своей природе современные телекоммуникационные системы – это компьютерные сети со сложной иерархической организацией телекоммуникационных протоколов. Поэтому в системе знаний и навыков специалистов по телекоммуникациям на первое место по приоритету становится знание телекоммуникационных протоколов, умение моделировать телекоммуникационные системы и сети. Другими словами, особое значение приобретают знания компьютерных сетей изнутри, то есть на уровне процессов, алгоритмов и протоколов. Возникают две проблемы, каким способом студентов с фактически нулевым уровнем программирования довести до уровня самостоятельного создания приложений, а в дальнейшем – научить программировать телекоммуникационные алгоритмы.

Анализ книг по программированию показывает, что большинство из них рассчитаны на подготовку профессиональных программистов. А вот как обучить создавать качественные программные продукты профессионалов-непрограммистов, например, профессионалов для телекоммуникаций, рекомендаций практически нет.

Формулирование цели статьи. При написании данной статьи ставилась цель – поделиться опытом подготовки специалистов по телекоммуникациям, сформулировать принципы и описать технику ускоренного обучения программированию профессионалов для телекоммуникаций на языке C++ в среде Borland C++ Builder.

Новизна статьи и новизна идей публикации. Материалы по данной тематике публикуются автором впервые. Данная статья является продолжением и конкретизацией идей, изложенных автором в предыдущих публикациях [1 – 3]. Работа впитала в себя 5-летний опыт преподавания дисциплины «Информатика» для студентов направления «Телекоммуникации». Как кажется автору, им изложен новый взгляд на технологию обучения специалистов для телекоммуникаций высокого профессионального уровня.

Результаты исследований

Подготовка к работе над первым проектом

В начале работы над первым проектом студентам объясняется следующее:

–- объектами в объектно-ориентированном программировании (ООП) являются абстрактные модели объектов реального мира;

– как и реальные объекты, объекты в ООП обладают тремя базовыми характеристиками – это: свойства, события, реакции на события;

– разрабатываемые приложения могут состоять из множества взаимодействующих объектов;

– разработка приложений состоит из визуального проектирования и собственно программирования; визуальное проектирование сводится к извлечению из библиотек готовых объектов, а программирование сводится к созданию программного кода реакций объектов на те или иные события;

– современные системы быстрого создания приложений до 95% кода создают сами, а остальные 5% создают программисты; поэтому разбираться во всех деталях создаваемого приложения непрофессионалу нет необходимости; это позволяет создавать качественные приложения, не вникая в детали;

© К.Н. Яковишин

– чтобы начинающему студенту создать первое качественное приложение, достаточно иметь добротный иллюстрированный самоучитель и путем «мавпування» (подражания) все действия, указанные в самоучителе, повторить в среде разработки приложений;

– точное повторение указаний из самоучителя в среде разработки приложений позволяет на подсознательном уровне (по аналогии с ускоренным изучением иностранных языков в режиме 25-го кадра) невероятно быстро осваивать технику создания приложений;

– все изложенное ниже полностью заимствовано из найденного в Интернете самоучителя [4]; автор самоучителя не известен, однако можно предположить, что самоучитель сделан на основе примера Культина Н.Б. «Сила тока», который содержится в приложении к [5] на CD-R.

Начало работы над первым проектом. Стартовая форма

Изучение возможностей построителя приложений C++ Builder, а также технологии визуального проектирования и событийного программирования можно начать с разработки приложения, которое вычисляет силу тока в электрической цепи. Сила тока вычисляется по известной формуле: I = U/R, где U – напряжение источника (вольт); R – величина сопротивления (Ом). Вид диалогового окна программы во время ее работы (после щелчка на кнопке **Вычислить**) приведен на рис. 1.

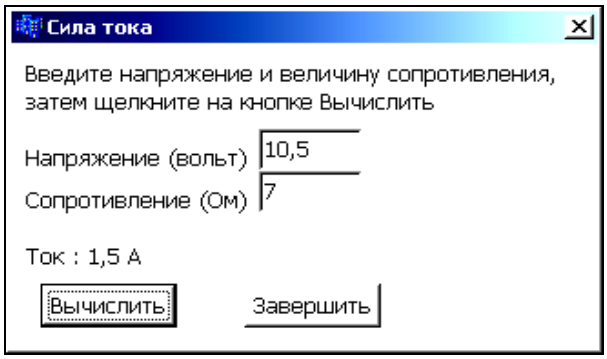

Рис. 1. Окно программы вычисления силы тока в электрической цепи (рис. 2.1 в [4])

Чтобы начать разработку нового приложения (так принято называть прикладные программы), надо запустить C++ Builder, в меню **File** выбрать команду **New | Application**.

Работа над новым проектом (так в C++ Builder называется разрабатываемое приложение) начинается с создания стартовой формы – главного окна программы. Стартовая форма создается путем изменения значений свойств формы **Form1** (настройки формы) и добавления к форме необходимых компонентов (объектов) (полей ввода, полей вывода текстовой информации, командных кнопок).

Для изменения значений свойств объектов, в том числе и формы, используется вкладка **Properties** (Свойства) диалогового окна **Object Inspector**. В левой колонке этой вкладки перечислены свойства выбранного объекта, в правой — указаны значения свойств.

При создании формы в первую очередь следует изменить значение свойства **Caption** (Заголовок). В нашем примере надо заменить текст **Form1** на **Сила тока**. Чтобы это сделать, нужно в окне **Object Inspector** щелкнуть левой кнопкой мыши в строке **Caption** (в результате будет выделено значение свойства и появится курсор) и ввести текст: сила тока (рис. 2.2 в [4]).

Здесь и в дальнейшем из-за ограниченного объема статьи приводятся ссылки на рисунки, а не сами рисунки.

Компоненты

Программа вычисления тока в электрической цепи должна получить от пользователя исходные данные – напряжение и величину сопротивления. Эти данные могут быть введены с клавиатуры в поля редактирования. Поэтому в форму надо добавить поле редактирования. Поля редактирования, поля вывода текста, списки, переключатели, командные кнопки и другие элементы пользовательского интерфейса называют компонентами.

Для того чтобы в форму разрабатываемого приложения добавить поле редактирования, надо в палитре компонентов, на вкладке **Standard**, щелкнуть на значке компонента **Edit** (рис. 2.6 в [4]), установить курсор в ту точку формы, в которой должен быть левый верхний угол компонента, и еще раз щелкнуть кнопкой мыши. В результате на форме появляется компонент **Edit** – поле редактирования (рис. 2.7 в [4]).

Каждому добавленному компоненту автоматически присваивается имя, которое состоит из названия компонента и его порядкового номера. Например, если к форме добавить два компонента **Edit**, то их имена будут **Edit1** и **Edit2**. Программист путем изменения значения свойства **Name** может изменить имя компонента. Однако в простых программах имена компонентов, как правило, не изменяют. Основные свойства компонента **Edit** приведены в табл. 2.3 [4].

Завершив работу над формой, можно приступить к созданию программного кода. Но перед этим рассмотрим два важных понятия: событие и функцию обработки события.

Событие и функция обработки события

Вид созданной формы подсказывает, как работает приложение. Очевидно, что пользователь должен ввести в поля редактирования исходные данные и щелкнуть мышью на кнопке **Вычислить**. Щелчок

на изображении командной кнопки – это пример того, что в Windows называется событием.

Событие (**Event**) – это то, что происходит во время работы приложения. В C++ Builder каждому событию присвоено имя. Например, щелчок кнопкой мыши – это событие **Onclick**, двойной щелчок мышью – событие **OnDblClick**.

Реакцией на событие должно быть какое-либо действие. В C++ Builder реакция на событие реализуется как функция обработки события. Таким образом, для того чтобы приложение выполняло некоторую работу в ответ на действия пользователя, программист должен написать функцию обработки соответствующего события. Следует обратить внимание на то, что значительную часть обработки событий берет на себя компонент. Поэтому программист должен разрабатывать функцию обработки события только в том случае, если реакция на событие отличается от стандартной или не определена. Например, если по условию задачи ограничений на символы, вводимые в поле **Edit**, нет, то процедуру обработки события **onKeypress** писать не надо, т.к. во время работы программы будет использована стандартная (скрытая от программиста) процедура обработки этого события.

Методику создания функций обработки событий рассмотрим на примере функции обработки события **onclick** для командной кнопки **Вычислить**.

Чтобы приступить к созданию функции обработки события, сначала надо выбрать компонент, для которого создается функция обработки события. Выбрать компонент можно в окне **Object Inspector** или щелчком на изображении компонента в форме. После этого в окне **Object Inspector** нужно выбрать вкладку **Events** (События).

В левой колонке вкладки **Events** (рис. 2.18 [4]) перечислены события, которые может воспринимать выбранный компонент (имя и тип компонента указаны в верхней части окна). Если для события определена функция обработки, то в правой колонке рядом с именем события будет выведено имя этой функции.

Для того чтобы создать функцию обработки события, нужно сделать двойной щелчок мышью в окне **Object Inspector**, в поле функции обработки соответствующего события (рис. 2.19 [4]). В результате этого откроется окно редактора кода, в которое будет добавлен шаблон функции обработки события, а в окне **Object Inspector** рядом с именем события появится сгенерированное C++ **Builder** имя функции обработки события (рис. 2.19 [4]).

C++ Builder присваивает функции обработки события имя, которое состоит из двух частей. Первая часть имени идентифицирует форму, содержащую объект (компонент), для которого создана процедура обработки события. Вторая часть имени идентифицирует сам объект и событие. В нашем примере имя формы – **Form1**, имя командной кнопки – **Button1**, а имя события – **Click**.

В окне редактора кода между фигурными скобками можно набирать инструкции, реализующие функцию обработки события.

В листинге 2.1 [4] приведен текст функции обработки события **onclick** для командной кнопки **Вычислить**. Обратите внимание на то, как представлена программа. Ее общий вид соответствует тому, как она выглядит в окне редактора кода: ключевые слова выделены полужирным шрифтом, комментарии – курсивом (выделение выполняет редактор кода). Кроме того, инструкции программы набраны с отступами в соответствии с принятыми в среде программистов правилами хорошего стиля.

Листинг 2.1 из [4]. Простейшая обработка события **onclick** на кнопке **Вычислить**

```
void__fastcall TForml::ButtonlClick(TObject *Sender)
{
float u; // напряжение
float r; // сопротивление
float i; // ток

// получить данные из полей ввода
u = StrToFloat(Editl->Text);
r = StrToFloat(Edit2->Text);

// вычислить  ток
i = u/r;

// вывести результат в поле метки
Label4->Caption = "Ток : " +
FloatToStrF(i,ffGeneral,7,2) + "A";
}
```

Функция **Button1Click** выполняет расчет силы тока и выводит результат расчета в поле **Label4**. Исходные данные вводятся из полей редактирования **Edit1** и **Edit2** путем обращения к свойству **Text**. Свойство **Text** содержит строку символов, которую ввел пользователь. Чтобы программа работала правильно, пользователь должен ввести в каждое поле редактирования целое или дробное число в правильном формате (при вводе дробного числа для разделения целой и дробной частей надо использовать запятую). Так как поле редактирования содержит текст (свойство **Text** строкового типа), необходимо выполнить преобразование строки в число. Эту задачу решает функция **strToFloat**, которой в качестве параметра передается содержимое поля редактирования — значение свойства **Text (Edit1->Text** — это содержимое поля **Edit1**). Функция **strToFloat** проверяет символы строки, переданной ей в качестве параметра, на допустимость и, если все символы верные, возвращает значение, соответствующее строке, полученной в качестве параметра.

После того как исходные данные будут помещены в переменные **u** и **r**, выполняется расчет. Вычисленная величина силы тока выводится в поле

Label4 путем присваивания значения свойству **Caption**. Для преобразования числа в строку символов (свойство **Caption** – строкового типа) используется функция **FloatToStrF**.

В листинге 2.2 [4] приведена процедура обработки события **onclick** на командной кнопке **Завершить**. Создается она точно так же, как и процедура обработки события **onclick** для командной кнопки Вычислить. В результате щелчка на кнопке Завершить программа должна завершить работу. Чтобы это произошло, надо закрыть окно программы. Делает это метод **close.**

Листинг 2.2. Процедура обработки события **Onclick** на кнопке **Завершить**

```
void __fastcall TForml::Button2Click(TObject*Sender)
{
Forml->Close();
}
```

Навигатор классов

Окно редактора кода разделено на две части (рис. 2.23 [4]). В правой части находится текст программы. Левая часть, которая называется навигатор классов (**ClassExplorer**), облегчает навигацию по тексту (коду) программы. В иерархическом списке, структура которого зависит от проекта, над которым идет работа, перечислены объекты проекта (формы и компоненты) и функции обработки событий. Выбрав элемент списка, можно быстро перейти к нужному фрагменту кода, например к функции обработки события.

Окно навигатора классов можно закрыть обычным образом. Если окно навигатора классов не доступно, то для того чтобы оно появилось на экране, нужно в меню **View** выбрать команду **ClassExplorer**.

Сохранение проекта

Проект – это набор файлов, используя которые компилятор создает выполняемый файл программы (exe-файл). В простейшем случае проект составляют: файл описания проекта (bpr-файл), файл главного модуля (cpp-файл), файл ресурсов (res-файл), файл описания формы (dfm-файл), заголовочный файл формы (h-файл) и файл описания функций формы (cpp-файл).

Чтобы сохранить проект, нужно в меню **File** выбрать команду **Save Project As**. Если проект еще ни разу не был сохранен, то C++ Builder сначала предлагает сохранить модуль (содержимое окна редактора кода) и поэтому на экране появляется окно **Save Unitl As**. В этом окне (рис. 2.29 [4]) надо выбрать папку, предназначенную для проектов, создать в ней папку для сохраняемого проекта, открыть ее и ввести имя модуля. В результате щелчка на кнопке **OK** в указанной папке будут созданы три файла: cpp, h и dfm, и на экране появится диалоговое окно Save Projectl As (рис. 2.30 [4]), в которое надо ввести имя проекта.

Следует обратить внимание, что имена файла модуля (**cpp**) и файла проекта (**bpr**) должны быть разными, т.к. C++ Builder в момент сохранения файла проекта создает одноименный cpp-файл (файл главного модуля). Кроме того, надо учесть, что имя генерируемого компилятором выполняемого файла совпадает с именем проекта. Поэтому файлу проекта следует присвоить такое имя, которое, по вашему мнению, должен иметь выполняемый файл программы, а файлу модуля – какое-либо другое имя, например, полученное путем добавления к имени проекта порядкового номера модуля.

Компиляция

Процесс преобразования исходной программы в выполняемую состоит из двух этапов: непосредственно компиляции и компоновки. На этапе компиляции выполняется перевод исходной программы в некоторое внутреннее представление. На этапе компоновки выполняется сборка (построение) программы.

После ввода текста функции обработки события и сохранения проекта можно, выбрав в меню **Project** команду **Compile**, выполнить компиляцию. Процесс и результат компиляции отражается в диалоговом окне **Compiling** (рис. 2.31 [4]). Если в программе нет синтаксических ошибок, то окно будет содержать сообщение: **Done: Compile Unit**, в противном случае будет выведено сообщение **Done: There are errors.**

В случае если компилятор обнаружит в программе ошибки и неточности, диалоговое окно **Compiling** будет содержать информацию о количестве синтаксических (**Errors**) и семантических (**Warnings**) ошибок, а также о числе подсказок (**Hints**). Сами сообщения об ошибках, предупреждения и подсказки находятся в нижней части окна редактора кода.

Чтобы перейти к фрагменту кода, который, по мнению компилятора, содержит ошибку, надо выбрать сообщение об ошибке (щелкнуть в строке сообщения левой кнопкой мыши) и из контекстного меню (рис. 2.32 [4]) выбрать команду **Edit Source**.

Процесс компиляции можно активизировать, выбрав в меню **Run** команду **Run**, которая запускает разрабатываемое приложение. Если будет обнаружено, что с момента последней компиляции в программу были внесены изменения или программа еще ни разу не компилировалась, то будет выполнена компиляция, затем – компоновка, и после этого программа будет запущена (естественно, только в том случае, если в программе нет ошибок).

Ошибки

Компилятор переходит ко второму этапу генерации выполняемой программы только в том случае,

если исходный текст не содержит синтаксических ошибок. В большинстве случаев в только что набранной программе есть ошибки. Программист должен их устранить. Процесс устранения ошибок носит итерационный характер. Обычно сначала устраняются наиболее очевидные ошибки, например, объявляются необъявленные переменные. После очередного внесения изменений в текст программы выполняется повторная компиляция. Следует обратить внимание на то, что компилятор не всегда может точно локализовать ошибку. Поэтому, анализируя фрагмент программы, который, по мнению компилятора, содержит ошибку, нужно обращать внимание не только на тот фрагмент кода, на который компилятор установил курсор, но и на тот, который находится в предыдущей строке. Например, в следующем фрагменте кода:

```
// вычислить ток
i = u/r
// вывести результат в поле метки
Label4->Caption = "Ток : " +
FloatToStrF(i,ffGeneral,7,2) + " A";
```

не поставлена точка с запятой после оператора присваивания. Компилятор это обнаруживает, выводит сообщение **statement missing ;**, но выделяет строку **Label4->caption = "Ток : " +** и устанавливает курсор после идентификатора **Label4**.

Компоновка

Если в программе нет ошибок, то можно выполнить компоновку. Для этого надо в меню Compile выбрать команду **Make** или **Build**. Разница между командами **Make** и **Build** заключается в следующем. Команда **Make** обеспечивает компоновку файлов проекта, а команда **Build** – принудительную перекомпиляцию, а затем – компоновку.

На этапе компоновки также могут возникнуть ошибки. Чаще всего причина ошибок во время компоновки состоит в недоступности файлов библиотек или других ранее откомпилированных модулей. Устраняются эти ошибки путем настройки среды разработки и включением в проект недостающих модулей. В простых проектах ошибки времени компиляции, как правило, не возникают.

Запуск программы

Пробный запуск программы можно выполнить непосредственно из среды разработки, не завершая работу с C++ Builder. Для этого нужно в меню **Run** выбрать команду **Run** или щелкнуть на командной кнопке **Run** (рис. 2.33 [4]).

Ошибки времени выполнения

Во время работы приложения могут возникать ошибки, которые называются ошибками времени выполнения (**run time errors**) или исключениями (**exceptions**). В большинстве случаев причинами исключений являются неверные исходные данные. Например, если во время работы программы вычисления силы тока в поле **Напряжение** ввести 10.5, т. е. разделить целую и дробную часть точкой, то в результате щелчка на кнопке **Вычислить** на экране появится окно с сообщением об ошибке (рис. 2.34 [4]).

Название программы

Название программы отображается во время ее работы в панели задач Windows, а также в заголовках окон сообщений, выводимых функцией **ShowMessage**. Название программы надо ввести в поле **Title** (рис. 2.38 [4]) вкладки **Application** диалогового окна **Project Options**, которое появляется в результате выбора в меню **Project** команды **Options**.

Структура простого проекта

Проект представляет собой набор программных единиц – модулей. Один из модулей, называемый главным, содержит инструкции, с которых начинается выполнение программы. Чтобы увидеть главный модуль, нужно в меню **Project** выбрать команду **View Source**. В качестве примера в листинге 2.5 [4] приведен текст главного модуля программы "Сила тока".

Листинг 2.5. [4] Главный модуль **(Amper.cpp)**

```
#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("Amper_1.cpp", Form1);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
        Application->Initialize();
        Application->CreateForm(__classid(TForm1), &Form1);
     Application->Run();
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    catch (...)
    {
        try
        {
            throw Exception("");
        }
        catch (Exception &exception)
        {
            Application->ShowException(&exception);
        }
    }
    return 0;
}
```

Начинается главный модуль директивами компилятору (точнее, препроцессору). Директива #include <vcl.h> информирует компилятор, что перед тем, как приступить непосредственно к компиляции, в текст главного модуля нужно включить заголовочный файл библиотеки визуальных компонентов — **vcl.h**.

Строка **USEFORM("Amper_1.cpp", Form1**) указывает, что в проект нужно включить файл модуля формы **Amper_1.cpp**, который содержит функции обработки событий для формы **Form1**. Далее следует описание главной функции программы – **winMain**. Функция **winMain** инициализирует внутренние структуры программы, создает форму **Form1** и запускает программу, что приводит к появлению на экране стартовой формы. Так как в проекте "Сила тока" только одна форма, то на экране именно она и появляется.

Инструкция обработки исключений **catch** выполняется, если в программе возникает ошибка. Таким образом, главный модуль обеспечивает вывод стартовой формы программы, дальнейшее поведение которой определяют функции обработки событий стартовой формы.

Помимо главного модуля в состав проекта входят модули формы. Для каждой формы C++ Builder создает отдельный модуль, который состоит из двух файлов: заголовочного файла и файла кода (содержимое этих файлов отражается в окне редактора кода). Заголовочный файл содержит описание формы (листинг 2.6 [4]), файл кода (модуль формы) — описание (текст) функций, в том числе и обработки событий (листинг 2.7 [4]).

Листинг 2.6 [4]. Заголовочный файл модуля формы **(Amper_1.h)**

```
#ifndef Amper_1H
#define Amper_1H
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Mask.hpp>
class TForm1 : public TForm
{
__published:       // IDE-managed Components
      TLabel *Label1;
      TLabel *Label2;
      TLabel *Label3;
      TEdit *Edit1;
      TEdit *Edit2;
      TButton *Button1;
      TButton *Button2;
      TLabel *Label4;
      void __fastcall Button1Click(TObject *Sender);
   void __fastcall Edit1KeyPress(TObject *Sender, char &Key);
   void __fastcall Button2Click(TObject *Sender);
   void __fastcall Edit2KeyDown(TObject *Sender, WORD &Key,
        TShiftState Shift);
   void __fastcall EditChange(TObject *Sender);
private:// User declarations
public:              // User declarations
      __fastcall TForm1(TComponent* Owner);
};
extern PACKAGE TForm1 *Form1;
#endif
```

Листинг 2.7 [4]. Модуль формы **(Amper_1.cpp)**

```
#include <vcl.h>
#pragma hdrstop
#include "Amper_l.h"
#ragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
__fastcall TForml::TForml(TComponent* Owner)
: TForm(Owner)
{
}
// щелчок на кнопке Вычислить
void__fastcall TForml::ButtonlClick(TObject *Sender)
{
float u; // напряжение
float r; // сопротивление
float i; // сила тока
// получить данные из полей ввода
// возможно исключение
try
{
u = StrToFloat(Editl->Text);
r = StrToFloat(Edit2->Text);
}
catch (EConvertError &e)
{
ShowMessage("При вводе дробных чисел исп..запятую.");
return;
}
// вычислить силу тока
// возможно исключение
 try
{
i = u/r;
}
catch (EZeroDivide &e)
{
ShowMessage("Сопротивление не должно быть равно нулю");
Edit1->SetFocus();  // курсор в поле Сопротивление
return;
 }
// вывести результат в поле метки Label4->Caption = "Ток : " +
FloatToStrF(i,ffGeneral,7,3);
}
// щелчок на кнопке Завершить
void__fastcall TForml::Button2Click(TObject *Sender)
{
Forml->Close(); // закрыть окно программы
}
```

Следует отметить, что значительное количество работы по генерации программного кода выполнил C++ Builder. Он полностью сформировал главный модуль **(Amper.cpp)**, заголовочный файл модуля формы **(Amper_1.h)**, значительную часть модуля формы **(Amper_1.cpp)**. Кроме того, C++ Builder, анализируя действия программиста, сформировал описание формы, файл проекта и файл ресурсов проекта.

Выводы

Профессионал по телекоммуникациям должен знать и уметь моделировать телекоммуникационные системы и протоколы в средах быстрого проектирования методами объектно-ориентированного программирования. Среда проектирования приложений Borland C++Builder 6 идеально подходит для этих

целей. Обучение программированию начинается с примера – первого проекта. Обучение осуществляется студентом самостоятельно с использованием иллюстрированного самоучителя. Студент то открывает, то закрывает окно с текстом самоучителя, шаг за шагом в точности повторяет все действия, описанные в самоучителе, в среде разработки приложений Borland C++Builder 6. Такое копирование (мавпування) фактически происходит на подсознательном уровне студента с очень высокой скоростью. Студент выполняет действия, смысл которых первоначально не понимает. Сознание практически отключено с момента начала и до конца процесса создания приложения. Даже результат работы уже созданного приложения после его запуска на выполнение также осуществляется на подсознательном уровне. Только после запуска приложения и оценки результатов его работы начинается второй этап обучения – осознание структуры проекта, осознание смысла действий, смысла операторов программного кода, перечень, содержание и размещение модулей, файлов проекта.

Главный вывод: за два месяца проектирования первого проекта студенты с нулевого уровня знаний и навыков по программированию достигают уровня самостоятельного создания приложений. В частности, студенты знают и умеют: запускать Построитель (Borland C++Builder 6), проектировать приложение из готовых объектов (путем простого перетаскивания объектов из библиотек), знают структуру приложений, в том числе модули и файлы приложения, главный модуль приложения, первую исполняемую инструкцию приложения, папки размещения файлов, формы (окна), объекты (компоненты), окна Построителя, свойства, события и реакции на события, основные шаги по созданию приложения (компиляция, компоновка, выполнение), возможные ошибки (синтаксические, семантические, времени выполнения) и способ их выявления, операторы препроцессора, операторы C++, функции в C++ (в том числе функции ввода-вывода).

Список литературы

1. Яковишин К.Н. VIP-специалисты для телекоммуникаций и IT-индустриии / К.Н Яковишин // Системи обробки інформації. – Х.: ХУ ПС, 2013. – Вип. 3(110), т. 2. – С. 164-168.

2. Яковишин К.Н. Новое в учебной практике моделирования телекоммуникационных систем / К.Н. Яковишин, Э.В. Скрипчинская // Тези науково-практичної конференції „Захист інформації в інформаційно-крмунікаційних системах". – К., 2013. – С. 83-86.

3. Yakovishin K.N. VIP-personnel for telecommunications / K.N. Yakovishin // Proceedings of the National Aviation University. – Kyiv. – 4'2013. – P. 161-164.

4. Иллюстрированный самоучитель по C++ Builder.

5. Культин Н.Б. C++ Builder в задачах и примерах / Н.Б. Культин. – СПб., 2005. – 336 с: ил.

Поступила в редколлегию 3.02.2015

Рецензент: д-р техн. наук, проф. Г.Ф. Конахович, Национальный авиационный университет, Киев.

ПРИСКОРЕНЕ НАВЧАННЯ ПРОГРАМУВАННЮ У СЕРЕДОВИЩІ BORLAND C++ BUILDER ПРОФЕСІОНАЛІВ ДЛЯ ТЕЛЕКОМУНІКАЦІЙ

К.М. Яковишин

Фахівець з телекомунікацій повинен знати і вміти моделювати телекомунікаційні системи і протоколи в середовищах швидкого проектування методами об'єктно-орієнтованого програмування. У зв'язку з цим розглядається техніка прискореного навчання програмуванню на мові C++ у середовищі Borland C++ Builder. У основі цієї техніки лежать наступні принципи. Навчання розпочинається з прикладу створення працюючого додатку шляхом "мавпування" (наслідування) дій, описаних в самовчителі. Використовується ідея сприйняття студентом знань і формування навичок на підсвідомому рівні (по аналогії з технікою навчання мовам в режимі 25-го кадру). Потім навчання триває проведенням все більш і більш детального аналізу структури проекту (додатка). Нарешті, на етапі контролю студенти успішно створюють новий власний додаток по аналогії з вивченим прикладом. В результаті за півтора-два місяці студенти з нульового рівня знань і навичок досягають рівня самостійного створення додатків.

***Ключові слова**: техніка прискореного навчання програмуванню, сприйняття знань і формування навичок на підсвідомому рівні, техніка навчання мовам в режимі 25-го кадру.*

SPEED-UP PROGRAMMING TRAINING IN BORLAND C++ BUILDER OF PROFESSIONALS FOR TELECOMMUNICATIONS

K.N. Yakovishin

Professional on telecommunications must know and able to design the telecommunication systems and protocols in the environments of the rapid planning by the methods of the object-oriented programming. In this connection, the technique of the speed-up educating to programming is examined in language C++ with in the environment of Borland C++ Builder. This technique next principles are the basis of. Educating is begun with the example of creation of working application the way of the "aping" (imitations) actions described in a manual for self-tuition. The idea of perception is used by the student of knowledge and forming of skills at subconscious level (by analogy with the technique of educating to the languages in the mode of 25th shot). Then educating proceeds realization more and more of the detailed analysis of structure of project (appendixes). Finally, on the stage of control students successfully create new own application by analogy with the studied example. As a result for one and a half-two months students from the zero level of knowledge and skills reach level independent creation of appendixes.

***Keywords**: technique of the speed-up educating to programming, perception of knowledge and forming skills at subconscious level, technician of educating to the languages in the mode of 25th shot.*