ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
ДИМИТРОГРАДСКИЙ ИНСТИТУТ ТЕХНОЛОГИЙ,
УПРАВЛЕНИЯ И ДИЗАЙНА

Лабораторная работа №1,2,3
по курсу: “ООП”
“*Преобразование типов. Дружественные функции. Конструктор копирования*”
Вариант №6

Выполнил студент гр. ВТ-31: Потеренко А.Г.
Проверил преподаватель: Наскальнюк А.Н.

Димитровград 2005 г.

Содержание.

Задание для работы.

Задание №1.
Определить два класса, строку с преобразование из ***char**** в строку и обратно, и целое ***long*** с преобразованием из ***long*** и обратно, а также взаимное преобразование ***String*** и ***long***.

Задание №2.
Создать класс комплексных чисел. Определить перегруженную функцию, возвращающую максимальный из двух аргументов. Функция не является членом класса комплексных чисел. Перегруженные функции имеют аргументы типа ***int***, ***double***, ***complex***. Тела перегруженных функций должны быть одинаковыми.

Задание №3.
Создать два класса вектор (float *) и матрица (float **). Определить конструкторы - по умолчанию, с параметром, для класса матрица с двумя параметрами, копирования, деструкторы. Определить функцию умножения матрицы на вектор.

Теория к заданиям.

№1.

Класс - это тип данных, определяемый пользователем. Понятия класса, структуры и объединения в C++ довольно близки друг к другу.

• Класс должен быть объявлен до того, как будет объявлена хотя бы одна переменная этого класса. Т.е. класс не может объявляться внутри объявления переменной.

• Синтаксис объявления класса следующий:

class <имя класса> : <список классов - родителей>

{

public: // доступно всем

<данные, методы, свойства, события>

__published // видны в Инспекторе Объекта и изменяемы

<данные, свойства>

protected: // доступно только потомкам

<данные, методы, свойства, события>

private: // доступно только в классе

<данные, методы, свойства, события>

} <список переменных>;

• Имя класса может быть любым допустимым идентификатором. Идентификаторы классов, наследующих классам библиотеки компонентов C++Builder, принято начинать с символа "T".

• Класс может наследовать поля (они называются данные-элементы), методы (они называются функции-элементы), свойства, события от других классов - своих предков, может отменять какие-то из этих элементов класса или вводить новые. Если предусматриваются такие классы-предки, то в объявлении класса после его имени ставится двоеточие и затем дается список родителей. В приведенном выше примере предусмотрено множественное наследование классам Class1 и Class2. Если среди классов-предков встречаются классы библиотеки компонентов C++Builder или классы, наследующие им, то множественное наследование запрещено.

• Если объявляемый класс не имеет предшественников, то список классов-родителей вместе с предшествующим двоеточием опускается.

• Доступ к объявляемым элементам класса определяется тем, в каком разделе они объявлены. Раздел public (открытый) предназначен для объявлений, которые доступны для внешнего использования. Это открытый интерфейс класса. Раздел __published (публикуемый) содержит открытые свойства, которые появляются в процессе проектирования на странице свойств Инспектора Объектов и которые, следовательно, пользователь может устанавливать в процессе проектирования. Раздел private (закрытый) содержит объявления полей и функций, используемых только внутри данного класса. Раздел protected (защищенный) содержит объявления, доступные только для потомков объявляемого класса. Как и в случае закрытых элементов, можно скрыть детали реализации защищенных элементов от конечного пользователя. Однако, в отличие от закрытых, защищенные элементы остаются доступны для программистов, которые захотят производить от этого класса производные классы, причем не требуется, чтобы производные классы объявлялись в этом же модуле.

• Перед именами классов-родителей в объявлении класса также может указываться спецификатор доступа (в примере public). Смысл этого спецификатора тот же, что и для элементов класса: при наследовании public (открытом наследовании) можно обращаться через объект данного класса к методам и свойствам классов-предков, при наследовании private подобное обращение невозможно. Подробнее этот вопрос рассмотрен в разделе наследование.

• По умолчанию в классах (в отличие от структур) предполагается спецификатор private. Поэтому можно включать в объявление класса данные и функции, не указывая спецификатора доступа. Все, что включено в описание до первого спецификатора доступа, считается защищенным. Аналогично, если не указан спецификатор перед списком классов-родителей, предполагается защищенное наследование.

• Объявления данных-элементов (полей) выглядят так же, как объявления переменных или объявления полей в структурах:

<тип> <имена полей>;

• В объявлении класса поля запрещается инициализировать. Для инициализации данных служат конструкторы.

• Объявления функций-элементов в простейшем случае не отличаются от обычных объявлений функций.

• После того, как объявлен класс, можно создавать объекты этого класса. Если ваш класс не наследует классам библиотеки компонентов C++Builder, то объект класса создается как любая переменная другого типа простым объявлением.

• В момент создания объекта класса, имеющего конструктор, можно инициализировать его данные, перечисляя в скобках после имени объекта значения данных.

• Создание переменных, использующих класс, можно совместить с объявлением самого класса, размещая их список между закрывающей класс фигурной скобкой и завершающей точкой с запятой.

• Создание объектов класса простым объявлением переменных возможно только в случае, если среди предков вашего класса нет классов библиотеки компонентов C++Builder. Если же такие предки есть, то создание указателя на объект этого класса возможно только операцией new.

№2.

Функция не член, получившая право доступа к закрытой части класса, называется другом класса (friend). Функция становится другом класса после описания как friend. Например:

```
class matrix;
class vector {
   float v[4];
   // ...
   friend vector multiply(matrix&, vector&);
};
class matrix {
   vector v[4];
   // ...
   friend vector multiply(matrix&, vector&);
};
```

Функция друг не имеет никаких особенностей, помимо права доступа к закрытой части класса. В частности, friend функция не имеет указателя this (если только она не является полноправным членом функцией). Описание friend - настоящее описание. Оно вводит имя функции в самой внешней области видимости программы и сопоставляется с другими описаниями этого имени. Описание друга может располагаться или в закрытой, или в открытой части описания класса; где именно, значения не имеет. Теперь можно написать функцию умножения, которая использует элементы векторов и матрицы непосредственно:

```
vector multiply(matrix& m, vector& v);
{
   vector r;
   for (int i = 0; i<3; i++) {    // r[i] = m[i] * v;
      r.v[i] = 0;
      for (int j = 0; j<3; j++)
         r.v[i] += m.v[i][j] * v.v[j];
   }
   return r;
}
```

Есть способы преодолеть эту конкретную проблему эффективности не используя аппарат friend (можно было бы определить операцию векторного умножения и определить multiply() с ее помощью). Однако существует много задач, которые проще всего решаются, если есть возможность предоставить доступ к закрытой части класса функции, которая не является членом этого класса.

№3.

Конструктор копирования – это специальный вид конструктора, получающий в качестве единственного параметра указатель на объект этого же класса:

T::T(const T&) {…/*Тело конструктора*/}

T – имя класса.

Этот конструктор вызывается в тех случаях, когда новый объект создается путем копирования существующего:

• При описании нового объекта с инициализацией другим объектом;

• При передаче объекта в функцию по значению;

• При возврате объекта из функции.

№4.

Конструктор класса - открытая функция-элемент, которая вызывается в момент создания объекта класса и должна инициализировать данные-элементы указанными в вызове значениями или значениями по умолчанию. Наличие конструктора в классе не обязательно. Но если конструктор отсутствует, то клиенты класса (внешние функции, использующие класс) должны сами заботиться об инициализации данных, т.е. о задании им некоторых начальных значений. Это не всегда возможно. Например, если класс имеет закрытые данные, предназначенные только для чтения, то для этих данных не предусматриваются открытые функции записи. И клиент не в состоянии присвоить данным какие-то начальные значения.

• Конструктор имеет то же имя, что и сам класс.

Пример объявления и реализации конструктора:

```
class MyClass
        {
        public:
          MyClass(void); // конструктор класса
          ...
        private:
          int A;
        ...
         };
        ...
        MyClass::MyClass(void) {A = 0;}
```

• В этом примере объявлен конструктор MyClass без параметров, который при создании объекта задает начальное значение поля A равным 0. Обратите внимание на то, что в отличие от других функций в объявлении конструктора не указывается тип возвращаемого значения.

• Простое задание в конструкторе значений данных в общем случае не гарантирует их целостность. Обычно нужна еще проверка допустимости данных. Например, если в классе есть функция записи SetA, осуществляющая такие проверки, то лучше обратиться к ней и при задании начального значения. В этом случае реализация конструктора может иметь вид:

MyClass::MyClass(void) { SetA(0); }

• Создание объекта описанного класса MyClass в программе должно осуществляться или объявлением соответствующей переменной:

MyClass MC;

или динамическим размещением переменной в памяти:

MyClass *PMC = new MyClass;

• В момент выполнения каждого из этих операторов неявным образом вызывается конструктор, устанавливающий начальные значения данных.

• Недостатком конструкторов показанного типа является то, что все начальные значения данных задаются в них конструктором. Вызывающая функция никак не может вмешаться в этот процесс и задать какое-то другое значение.

• Другой крайностью являются конструкторы, в которых все начальные значения задаются как параметры. Например, прототип конструктора может иметь вид

MyClass(int);

а его реализация:

MyClass::MyClass(int a) { SetA(a); }

В этом случае поле FA инициализируется параметром, передаваемым в конструктор. Создание объекта подобного класса должно выполняться операторами

MyClass MC(1);

или

MyClass *PMC = new MyClass(1);

в которых подразумевается, что начальное значение поля FA должно быть равно 1.

• Такой конструктор обычно тоже неудобен, поскольку в классе может быть много параметров и задавать значения их всех при создании объекта очень громоздко и чревато ошибками.

• Чаще всего используются конструкторы с параметрами по умолчанию. В этом случае объявление конструктора может иметь вид:

MyClass(int = 0);

а его реализация:

MyClass::MyClass(int a) { SetA(a); }

• Объект такого класса можно создавать любым из приведенных ранее операторов создания объекта. Если при создании указывается аргумент, то его значение присваивается полю. Если аргумент не указывается, то присваивается значение по умолчанию (в нашем примере 0). Этот вариант конструктора наиболее гибкий. Поэтому он чаще всего используется при построении классов.

• В объявлении класса могут быть определены не только поля переменных, но и некоторые именованные константы. Например:

const int MaxA;

Подобная константа может служить, в частности, предельно допустимым значением поля FA.

• Если такая константа объявлена как статическая, то в ее объявление в классе можно непосредственно включить инициализацию:

static const int MaxA = 10;

Но тогда это значение клиент при желании не сможет изменить. А задать значение такой константы в конструкторе невозможно, поскольку компилятор не разрешает присваивать значения константам. Выходом из положения является специальный синтаксис конструктора с инициализатором элементов. Инициализатор элементов записывается после заголовка конструктора в его реализации, предваряется двоеточием и содержит имена константных данных, после которых в скобках указываются их значения. Например, если в объявлении вашего класса MyClass имеются строки

const int MaxA;

const int MinA;

вводящие две константы - максимальное и минимальное значения переменной A, то реализацию конструктора такого класса с описанным ранее прототипом

MyClass(int = 0);

надо дополнить инициализатором элементов:

MyClass::MyClass(int a) : MaxA(10), MinA(1) {SetA(a);};

В данном случае инициализатор задает константе MaxA начальное значение 10, а константе MinA - значение 1.

• Можно предоставить пользователю возможность изменять значения констант в момент создания объекта. В этом случае в конструкторе с умолчанием надо предусмотреть для констант соответствующие значения по умолчанию:

MyClass(int A = 0, int MaxA = 10, int MinA = 1);

или

MyClass(int = 0, int = 10, int = 1);

(второй вариант менее удобен, так как не позволяет по прототипу функции понять, в какой последовательности должны задаваться параметры).

• Теперь остановимся на деструкторах. Это специальные функции-элементы, срабатывающие при уничтожении динамически размещенного объекта класса и освобождающие занимаемую им память. Имя деструктора совпадает с именем класса, но перед ним записывается символ тильда "~". Как и для конструктора, в деструкторе не указывается возвращаемый тип. Например:

```
class MyClass
                    {
                    public:
                      ~MyClass(); // деструктор класса
                      ...
                    };
```

Деструкторы необходимы, если конструктор или какие-то функции-элементы класса динамически распределяют память, создавая в ней какие-то объекты. Тогда деструктор должен эти объекты удалять. В остальных случаях можно обычно обойтись без деструктора.

• Если деструктор явным образом в классе не объявлен, компилятор сам генерирует необходимые коды освобождения памяти.

№5.

C++ позволяет определить несколько функций с одним и тем же именем, если эти функции имеют разные наборы параметров (по меньшей мере разные типы параметров). Эта особенность называется ***перегрузкой функции***. При вызове перегруженной функции компилятор C++ определяет соответствующую функцию путем анализа количества, типов и порядка следования аргументов в вызове. Перегрузка функции обычно используется для создания нескольких функций с одинаковым именем, предназначенных для выполнения сходных задач, но с разными типами данных. Применение при этом перегруженных функций делает программы более понятными и легко читаемыми.

Пусть, например, вы хотите определить функции, добавляющие в заданную строку типа (char *) символ пробела и значение целого числа, или значение числа с плавающей запятой, или значение булевой переменной. Причем хотите обращаться в любом случае к функции, которую называете, например, ToS, предоставив компилятору самому разбираться в типе параметра и в том, какую из функций надо вызывать в действительности. Для решения этой задачи вы можете описать следующие функции:

```
char * ToS(char *S,int X)
{return strcat(strcat(S," "),IntToStr(X).c_str());}
char * ToS(char *S, double X)
{return strcat(strcat(S," "),FloatToStr(X).c_str());}
char * ToS(char *S, bool X)
{if (X) return strcat(S," true");
 else return strcat(S," false");}
```

Тогда в своей программе вы можете написать, например, вызовы:

char S[128] = "Значение =";

char S1 = ToS(S,5);

или

char S[128] = "Значение =";

char S2 = ToS(S,5.3);

или

char S[128] = "Значение =";

char S3 = ToS(S,true);

В первом случае будет вызвана функция с целым аргументом, во втором - с аргументом типа double, в третьем - с булевым аргументом. Вы видите, что перегрузив соответствующие функции вы существенно облегчили свою жизнь, избавившись от необходимости думать о типе параметра.

Приведем еще один пример, в котором перегруженные функции различаются количеством параметров. Ниже описана перегрузка функции, названной Area и вычисляющей площадь круга по его радиусу R, если задан один параметр, и площадь прямоугольника по его сторонам a и b, если задано два параметра:

double Area(double R) { return 6.28318 * R * R; }

double Area(double a, double b) { return a * b; }

Тогда операторы вида

S1 = Area(1);

S2 = Area(1,2);

приведут в первом случае к вызову функции вычисления площади круга, а во втором - к вызову функции вычисления площади прямоугольника.

Перегруженные функции различаются компилятором с помощью их сигнатуры - комбинации имени функции и типов ее параметров. Компилятор кодирует идентификатор каждой функции по числу и типу ее параметров (иногда это называется декорированием имени), чтобы иметь возможность осуществлять надежное связывание типов. Надежное связывание типов гарантирует, что вызывается надлежащая функция и что аргументы согласуются с параметрами. Компилятор выявляет ошибки связывания и выдает сообщения о них.

Для различения функций с одинаковыми именами компилятор использует только списки параметров. Перегруженные функции не обязательно должны иметь одинаковое количество параметров.

Программисты должны быть осторожными, имея дело в перегруженных функциях с параметрами по умолчанию, поскольку это может стать причиной неопределенности. Функция с пропущенными аргументами по умолчанию может оказаться вызванной аналогично другой перегруженной функции; это синтаксическая ошибка.

Рассмотренный аппарат перегрузки функций - только один из возможных способов решения поставленной задачи, правда, универсальный, позволяющий работать и с разными типами параметров, и с разным числом параметров. В следующем разделе рассмотрен еще один механизм - шаблоны, позволяющий решать аналогичные задачи, правда, для более узких классов функций.

№6.

Все операции C++ могут быть перегружены, кроме операций точка **(.)**, разыменование **(*)**, разрешение области действия **(::)**, условная **(?:)** и **sizeof**.

Операции =, [ ], ( ) и -> могут быть перегружены только как нестатические функции-элементы. Они не могут быть перегружены для перечислимых типов.

Все остальные операции можно перегружать, чтобы применять их к каким-то новым типам объектов, вводимым пользователем. Кроме того, многие операции уже перегружены в C++. Например, арифметические операции применяются к разным типам данных - целым числам, действительным и т.д., именно в результате того, что они перегружены.

Операции перегружаются путем составления описания функции (с заголовком и телом), как это делается для любых функций, за исключением того, что в этом случае имя функции состоит из ключевого слова ***operator***, после которого записывается перегружаемая операция. Например, имя функции ***operator+*** можно использовать для перегрузки операции сложения.

Чтобы использовать операцию над объектами классов, эта операция должна быть перегружена, но есть два исключения. Операция присваивания (=) может быть использована с каждым классом без явной перегрузки. По умолчанию операция присваивания сводится к побитовому копированию данных-элементов класса. Такое побитовое копирование опасно для классов с элементами, которые указывают на динамически выделенные области памяти; для таких классов следует явно перегружать операцию присваивания. Операция адресации (&) также может быть использована с объектами любых классов без перегрузки; она просто возвращает адрес объекта в памяти. Но операцию адресации можно также и перегружать.

Перегрузка не может изменять старшинство и ассоциативность операций. Нельзя также изменить число операндов операции. Например, унарную операцию можно перегрузить только как унарную.

Перегрузка больше всего подходит для математических классов. Они часто требуют перегрузки значительного набора операций, чтобы обеспечить согласованность со способами обработки этих математических классов в реальной жизни. Например, было бы странно перегружать только сложение класса комплексных чисел, потому что обычно с комплексными числами используются и другие арифметические операции.

Цель перегрузки операций состоит в том, чтобы обеспечить такие же краткие выражения для типов, определенных пользователем, какие C++ обеспечивает с помощью богатого набора операций для встроенных типов. Однако, перегрузка операций не выполняется автоматически; чтобы выполнить требуемые операции, программист должен написать функции, осуществляющие перегрузки операций.

Алгоритм работы программ.

Алгоритм задания №1.

Создаем класс **strin** – класс преобразования из **AnsiString** в **char*** и обратно**.** Методы этого классы занимаются данным преобразованием.

Создаем класс **INTSTR** – класс преобразования из **AnsiString** в **long** и обратно**.** Методы этого классы занимаются данным преобразованием.

Создаем экземпляры классов и вызываем соответствующие методы.

Алгоритм задания №2.

Объявляем перегруженные функции:

```
//--------------------Перегруженная функция №1-----------------------
float FUNC1(int x,int y);
float FUNC2(int x,int y);
float FUNC3(int x,int y);
//--------------------Перегруженная функция №2-----------------------
double FUNC1(double x,double y);
double FUNC2(double x,double y);
double FUNC3(double x,double y);
//--------------------Перегруженная функция №3-----------------------
double FUNC1(complex <float> t);
double FUNC2(complex <float> t);
double FUNC3(complex <float> t);
```

Объявляем класс комплексных чисел: **COMP.** Объявляем в этом же классе дружественные функции.

Создаем экземпляр класса комплексных чисел: **CO1**.

Пользователь вводит аргументы действительной и мнимой части. Затем с помощью дружественных функций вычисляются необходимые значения.
FUNC1 – вычисляет arg(z), где z – комплексное число.
FUNC2 – вычисляет abs(z), где z – комплексное число.
FUNC3 – вычисляет max или min аргументов z,где z – комплексное число.

Алгоритм задания №3.

Создадим три класса: **VECTOR**, **MATRICA, CLASSVECTORS**. Для трех классов создадим конструктор по умолчанию (параметр равен 10). Конструктор копирования, деструктор. Процедура:

float show(int i);

служит для вывода элементов вектора. Процедура:

void PROCPOLE(int l); //l - порядок вектора

служит для создания динамической структуры. Процедура:

int SHOWPOLE();

служит для показа порядка динамической структуры (если все-таки пользователь ввел порядок). Процедура:

void POLES(int r);

изменяет значение по умолчанию для данного экземпляра. После создания нового – параметр по умолчанию восстанавливается.

Указатель на динамическую структуру:

float *vect; //vect - указатель на вещественный тип

Для динамической матрицы:

float **matr; //matr - указатель на вещественный тип

Из всех алгоритмов интересен алгоритм создания динамической матрицы:

1) Создаем указатель matr на массив указателей

```
matr = new float *[l];
```

2) Проходим каждый элемент массива указателей для создания в них указателей на массив вещественных чисел:

```
for (int i=0;i<l;i++)
   matr[i]= new float [l];   //matr[i] - массив вещественных чисел
```

3) Заполняем генератором случайных чисел элементы данной матрицы.

При удалении данной матрицы из памяти мы сначала освобождаем память, занимаемую массивом чисел, затем удаляем сам указатель на массив указателей.

Класс "результат" содержит произведение вектора и матрицы. Конкретно: VECTORS – указатель на массив вещественных чисел, именно здесь будет хранится результат. Методы данного класса почти одинаковы, за исключением того, что мы перегрузили оператор присваивания.

```
CLASSVECTORS operator =(float *M)
 {
   for (int i=0;i<CLASSVECTORS::SHOWPOLE();i++)
    {
     CLASSVECTORS::POLE(i,M[i]);
    };
 }
};
```

В не классов перегрузили оператор умножения:

```
float* operator *(VECTOR &M1, MATRICA &M2)
{
  float *mas;
  mas = new float [M1.SHOWPOLE()];
  float SUM;
  for (int i=0;i<M1.SHOWPOLE();i++)
     {
      SUM=0;
      for (int j=0;j<M1.SHOWPOLE();j++)
        {
         SUM=SUM+M1.show(j)*M2.show(i,j);
        };
      mas[i]=SUM;
     };
  return (mas);
};
```

Здесь функция возвращает указатель на массив вещественных чисел.

Листинги.

Листинг задания №1.

Project1.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("Unit1.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;
}
//---------------------------------------------------------------------------
```

Unit1.h

```
//---------------------------------------------------------------------------
#include <Forms.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Classes.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
   __published:
        TButton *Button1;
        void __fastcall Button1Click(TObject *Sender);
     public:
        __fastcall TForm1(TComponent* Owner);
};
```

Unit1.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include "Unit1.h"
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner): TForm(Owner){}
//---------------------------------------------------------------------------
//------------------------Первый класс---------------------------------------
//---------------------------------------------------------------------------
class strin
  {
    public:
     char * METOD_ANSI_IN_CHAR (AnsiString STRIN);
     AnsiString METOD_CHAR_IN_ANSI (char * chr);
    private:
     AnsiString STR;
  };
//---------------------------------------------------------------------------
AnsiString strin::METOD_CHAR_IN_ANSI (char * chr)
{
 STR=chr;
 return (STR);
```

```
}
//---------------------------------------------------------------------------
char * strin::METOD_ANSI_IN_CHAR (AnsiString STRIN)
{
 return (STRIN.c_str());
}
//---------------------------------------------------------------------------
//----------------------Класс взаимного преобразования-----------------------
//---------------------------------------------------------------------------
class INTSTR
  {
    public:
      long METOD_CHAR_IN_LONG (char * chr);
      char * METOD_LONG_IN_CHAR (long lon);
      AnsiString METOD_LONG_IN_ANSI (long lon);
      long METOD_ANSI_IN_LONG (AnsiString RT);
    private:
      AnsiString ST;
      long lon;
      char * chr;
  };
//---------------------------------------------------------------------------
long INTSTR::METOD_CHAR_IN_LONG (char * chr)
{
 return (atoi(chr));
};
//---------------------------------------------------------------------------
char * INTSTR::METOD_LONG_IN_CHAR (long lon)
{
 chr=""; //Инициализируем указатель на строку
 itoa(lon,chr,10);
 return (chr);
};
//---------------------------------------------------------------------------
AnsiString INTSTR::METOD_LONG_IN_ANSI (long lon)
{
 ST=lon;
 return (ST);
};
//---------------------------------------------------------------------------
long INTSTR::METOD_ANSI_IN_LONG (AnsiString RT)
{
 lon=StrToInt(RT);
 return (lon);
};
//---------------------------------------------------------------------------
//----------------------Выполняем все преобразования-------------------------
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 //----------------------Преобразования первого класса-------------------------
 strin STR;
 AnsiString STI="";
 char * b="";
 STI=STR.METOD_CHAR_IN_ANSI(b); //Неявное преобразование ST=b
 b=STR.METOD_ANSI_IN_CHAR(STI); //Неявное преобразование b=ST
 //-----------------------Взаимное преобразование-------------------------------
 INTSTR INS;
 long LONG=0;
 b=INS.METOD_LONG_IN_CHAR(LONG);
 LONG=INS.METOD_CHAR_IN_LONG(b);
 STI=INS.METOD_LONG_IN_ANSI(LONG);
 LONG=INS.METOD_ANSI_IN_LONG(STI);
}
//---------------------------------END-----------------------------------------
```

Листинг задания №2.

Project1.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("Unit1.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;
}
//---------------------------------------------------------------------------
```

Unit1.h

```
#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include "DynamicSkinForm.hpp"
#include "SkinData.hpp"
#include "SkinBoxCtrls.hpp"
#include "SkinCtrls.hpp"
#include <Mask.hpp>
#include <complex>
//---------------------------------------------------------------------------
using namespace std;
//--------------------------Класс TForm1-------------------------------------
class TForm1 : public TForm
{
__published:
        TspDynamicSkinForm *spDynamicSkinForm1;
        TspSkinData *spSkinData1;
        TspSkinStdLabel *spSkinStdLabel1;
        TspSkinEdit *spSkinEdit1;
        TspSkinStdLabel *spSkinStdLabel2;
        TspSkinEdit *spSkinEdit2;
        TspSkinGroupBox *spSkinGroupBox1;
        TspSkinButton *spSkinButton1;
        TspSkinButton *spSkinButton2;
        TspSkinEdit *spSkinEdit3;
        TspSkinStdLabel *spSkinStdLabel3;
        TspSkinButton *spSkinButton3;
        TspSkinEdit *spSkinEdit4;
        TspSkinStdLabel *spSkinStdLabel4;
        TspSkinStdLabel *spSkinStdLabel5;
        TspSkinEdit *spSkinEdit5;
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall FORMSS(TObject *Sender, int &NewWidth,
                               int &NewHeight, bool &Resize);
        void __fastcall spSkinButton1Click(TObject *Sender);
        void __fastcall spSkinButton2Click(TObject *Sender);
        void __fastcall spSkinButton3Click(TObject *Sender);
public:
        __fastcall TForm1(TComponent* Owner);
        //---------------------Перегруженная функция №1-------------------------
        float FUNC1(int x,int y);
        float FUNC2(int x,int y);
        float FUNC3(int x,int y);
        //---------------------Перегруженная функция №2-------------------------
        double FUNC1(double x,double y);
        double FUNC2(double x,double y);
        double FUNC3(double x,double y);
        //---------------------Перегруженная функция №3-------------------------
        double FUNC1(complex <float> t);
        double FUNC2(complex <float> t);
        double FUNC3(complex <float> t);
        //----------------------Класс комплексных чисел-------------------------
```

```
        class COMP
           {
             private:
               complex <float> z;
               friend float TForm1::FUNC1(int x,int y);
               friend float TForm1::FUNC2(int x,int y);
               friend float TForm1::FUNC3(int x,int y);
               //---------------------------------------------------------------
               friend double TForm1::FUNC1(double x,double y);
               friend double TForm1::FUNC2(double x,double y);
               friend double TForm1::FUNC3(double x,double y);
               //---------------------------------------------------------------
               friend double TForm1::FUNC1(complex <float> t);
               friend double TForm1::FUNC2(complex <float> t);
               friend double TForm1::FUNC3(complex <float> t);
           };
        //----------------------Создаем экземпляр класса-------------------------
        COMP CO1;
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
#endif
```

Unit1.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "DynamicSkinForm"
#pragma link "SkinData"
#pragma link "SkinBoxCtrls"
#pragma link "SkinCtrls"
#pragma resource "*.dfm"
//---------------------------------------------------------------------------
TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner) {}
//-------------------------Когда изменяем размеры формы-----------------------
void __fastcall TForm1::FORMSS(TObject *Sender, int &NewWidth,
      int &NewHeight, bool &Resize)
{
  Form1->spSkinGroupBox1->Width=Form1->Width-26;
  Form1->spSkinGroupBox1->Height=Form1->Height-107;
  Form1->spSkinEdit1->Width=Form1->Width-202;
  Form1->spSkinEdit2->Width=Form1->Width-202;
  Form1->spSkinEdit3->Width=Form1->Width-290;
  Form1->spSkinEdit4->Width=Form1->Width-290;
  Form1->spSkinEdit5->Width=Form1->Width-290;
}
//-----------------------------Инициализация-----------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Form1->Width=459;
  Form1->Height=239;
  Form1->Top=176;
  Form1->Left=294;
  Form1->spSkinData1->LoadFromCompressedFile("MacRemix.skn");
}
//---------------------------Первая функция (int)------------------------------
float TForm1::FUNC1(int x, int y)
{
 try
 {
  Form1->CO1.z._M_re=x;
  Form1->CO1.z._M_im=y;
  return(arg(Form1->CO1.z));
 }
 catch (...)
 {};
}
float TForm1::FUNC2(int x, int y)
{
 try
 {
  Form1->CO1.z._M_re=x;
  Form1->CO1.z._M_im=y;
  return(abs(Form1->CO1.z));
 }
 catch (...)
 {};
```

```
}
float TForm1::FUNC3(int x, int y)
{
 try
  {
   Form1->CO1.z._M_re=x;
   Form1->CO1.z._M_im=y;
   if (Form1->CO1.z._M_re > Form1->CO1.z._M_im) return(Form1->CO1.z._M_re);
   if (Form1->CO1.z._M_re < Form1->CO1.z._M_im) return(Form1->CO1.z._M_im);
   else return(0);
  }
 catch (...)
 {};
}
//----------------------------Вторая функция (double)---------------------------
double TForm1::FUNC1(double x, double y)
{
 Form1->CO1.z._M_re=x;
 Form1->CO1.z._M_im=y;
 return(arg(Form1->CO1.z));
}
double TForm1::FUNC2(double x, double y)
{
 Form1->CO1.z._M_re=x;
 Form1->CO1.z._M_im=y;
 return(abs(Form1->CO1.z));
}
double TForm1::FUNC3(double x, double y)
{
 Form1->CO1.z._M_re=x;
 Form1->CO1.z._M_im=y;
 if (Form1->CO1.z._M_re > Form1->CO1.z._M_im) return(Form1->CO1.z._M_re);
 if (Form1->CO1.z._M_re < Form1->CO1.z._M_im) return(Form1->CO1.z._M_im);
 else return(0);
}
//----------------------------Третья функция (complex)---------------------------
double TForm1::FUNC1(complex <float> t)
{
 Form1->CO1.z._M_re=t._M_re;
 Form1->CO1.z._M_im=t._M_im;
 return(arg(Form1->CO1.z));
}
double TForm1::FUNC2(complex <float> t)
{
 Form1->CO1.z._M_re=t._M_re;
 Form1->CO1.z._M_im=t._M_im;
 return(abs(Form1->CO1.z));
}
double TForm1::FUNC3(complex <float> t)
{
 Form1->CO1.z._M_re=t._M_re;
 Form1->CO1.z._M_im=t._M_im;
 if (Form1->CO1.z._M_re > Form1->CO1.z._M_im) return(Form1->CO1.z._M_re);
 if (Form1->CO1.z._M_re < Form1->CO1.z._M_im) return(Form1->CO1.z._M_im);
 else return(0);
}
//---------------------------------------------------------------------------
//----------------------------Обработчики событий--------------------------------
//---------------------------------------------------------------------------
void __fastcall TForm1::spSkinButton1Click(TObject *Sender)
{
 try
 {
  int a,b;
  a=StrToInt(spSkinEdit1->Text);
  b=StrToInt(spSkinEdit2->Text);
  if (a!=0 || b!=0)
  {
    spSkinEdit3->Text=VarToStr(Form1->FUNC1(a,b));
    spSkinEdit4->Text=VarToStr(Form1->FUNC2(a,b));
    if (Form1->FUNC3(a,b)!=0)
      spSkinEdit5->Text="Максимальный аргумент:" +VarToStr(Form1->FUNC3(a,b));
    else
      spSkinEdit5->Text="Аргументы равны";
  }
 }
 catch (...)
 {
  ShowMessage("Ошибка ввода/вывода");
 };
}
//---------------------------------------------------------------------------
void __fastcall TForm1::spSkinButton2Click(TObject *Sender)
```

```
{
 try
 {
  double a,b;
  a=StrToFloat(spSkinEdit1->Text);
  b=StrToFloat(spSkinEdit2->Text);
  if (a!=0 || b!=0)
   {
    spSkinEdit3->Text=VarToStr(Form1->FUNC1(a,b));
    spSkinEdit4->Text=VarToStr(Form1->FUNC2(a,b));
    if (Form1->FUNC3(a,b)!=0)
      spSkinEdit5->Text="Максимальный аргумент:" +VarToStr(Form1->FUNC3(a,b));
    else
      spSkinEdit5->Text="Аргументы равны";
   };
 }
 catch (...)
 {
  ShowMessage("Ошибка ввода/вывода");
 };
}
//---------------------------------------------------------------------------
void __fastcall TForm1::spSkinButton3Click(TObject *Sender)
{
 try
 {
  complex <float> p;
  p._M_re=StrToFloat(spSkinEdit1->Text);
  p._M_im=StrToFloat(spSkinEdit2->Text);
  if ( p._M_re!=0 || p._M_im!=0)
    {
     spSkinEdit3->Text=VarToStr(Form1->FUNC1(p));
     spSkinEdit4->Text=VarToStr(Form1->FUNC2(p));
     if (Form1->FUNC3(p._M_re,p._M_im)!=0)
       spSkinEdit5->Text="Максимальный аргумент:" +VarToStr(Form1->FUNC3(p));
     else
       spSkinEdit5->Text="Аргументы равны";
    };
 }
 catch (...)
 {
  ShowMessage("Ошибка ввода/вывода");
 };
}
//---------------------------------------------------------------------------
```

Листинг задания №3.

Project1.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("Unit1.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;
}
//---------------------------------------------------------------------------
```

Unit1.h

```
//-----------------------------------OK--------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include "SkinCtrls.hpp"
#include "SkinData.hpp"
#include "SkinBoxCtrls.hpp"
#include <Mask.hpp>
#include <ComCtrls.hpp>
#include "SkinTabs.hpp"
#include "DynamicSkinForm.hpp"
//---------------------------------------------------------------------------
//----------------------Класс ВЕКТОР-----------------------------------------
//---------------------------------------------------------------------------
const CONSTT=10;
class VECTOR
   {
     public:
        //----------------Конструктор по умолчанию с параметром-----------------
        VECTOR (int=CONSTT); //CONSTT - число элементов в векторе "по умолчанию"
        //------------Конструктор копирования-------------------------------
        VECTOR::VECTOR(const VECTOR &V);
        //----------------------------Деструктор-------------------------------
        VECTOR::~VECTOR();
        //---------------------------------------------------------------------
        float show(int i); //Процедура показа элементов вектора
        //---------Процедура создания динамического массива---------------------
        void PROCPOLE(int i); //i - порядок вектора
        //--------------Показываем порядок вектора-----------------------------
        int SHOWPOLE();
        //-----------------Изменение pole-------------------------------------
        void POLES(int i);
     private:
        int pole; //pole - порядок вектора
        float *vect; //vect - указатель на вещественный тип
   };
//---------------------------------------------------------------------------
//----------------------Класс МАТРИЦА----------------------------------------
//---------------------------------------------------------------------------
class MATRICA
   {
     public:
        //----------------Конструктор по умолчанию с параметром-----------------
        MATRICA (int = CONSTT);//CONSTT-число элементов в векторе "по умолчанию"
        //------------Конструктор копирования-------------------------------
        MATRICA::MATRICA(const MATRICA &M);
        //----------------------------Деструктор-------------------------------
        MATRICA::~MATRICA();
        //---------------------------------------------------------------------
        float show(int i, int j); //Процедура показа элементов матрицы
        //---------Процедура создания динамической матрицы----------------------
        void PROCPOLE(int i); //i - порядок матрицы
        //--------------Показываем порядок матрицы-----------------------------
        int SHOWPOLE();
        //-----------------Изменение pole-------------------------------------
        void POLES(int i);
     private:
        int pole; //pole - порядок матрицы
        float **matr; //matr - указатель на вещественный тип
   };
//---------------------------------------------------------------------------
//----------------------Класс Результат--------------------------------------
//---------------------------------------------------------------------------
class CLASSVECTORS
  {
    private:
     int pole; //pole - порядок вектора
     float *VECTORS; //Здесь хранится результат умножения вектора и матрицы
    public:
     //--------------------Конструктор по умолчанию с параметром----------------
     CLASSVECTORS (int = CONSTT); //CONSTT - число элементов "по умолчанию"
     //-------------------------------Деструктор-----------------------------
     CLASSVECTORS::~CLASSVECTORS();
     //-------------Обращение к элементам вектора-----------------------------
     float show(int k);  //Показываем значение поля
     void POLE(int k, int SUM); //Заносим новое значение в вектор
     //------------Процедура создания динамического массива--------------------
     void PROCPOLE(int i); //i - порядок вектора
     //-----------------Показываем порядок вектора-----------------------------
```

```
     int SHOWPOLE();
     //-----------------Изменение pole-----------------------------------------
     void POLES(int i);
     //---------------Оператор присваивания----------------------------------
     CLASSVECTORS operator =(float *M)
       {
          for (int i=0;i<CLASSVECTORS::SHOWPOLE();i++)
           {
             CLASSVECTORS::POLE(i,M[i]);
           };
       }
  };
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:                        // IDE-managed Components
        TspDynamicSkinForm *spDynamicSkinForm1;
        TspSkinData *spSkinData1;
        TspSkinGroupBox *spSkinGroupBox1;
        TspSkinPageControl *spSkinPageControl1;
        TspSkinTabSheet *spSkinTabSheet1;
        TspSkinTabSheet *spSkinTabSheet2;
        TspSkinTabSheet *spSkinTabSheet4;
        TspSkinStdLabel *spSkinStdLabel1;
        TMemo *Memo1;
        TspSkinStdLabel *spSkinStdLabel3;
        TMemo *Memo3;
        TspSkinStdLabel *spSkinStdLabel2;
        TspSkinEdit *spSkinEdit2;
        TMemo *Memo2;
        TspSkinButton *spSkinButton1;
        TMemo *Memo4;
        TMemo *Memo5;
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall spSkinButton1Click(TObject *Sender);
        void __fastcall FORMRES(TObject *Sender, int &NewWidth,
          int &NewHeight, bool &Resize);
private:                            // User declarations
public:                                     // User declarations
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
```

Unit1.cpp

```
////////////////////////////OK/////////////////////////////////////////////////
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include <math.h>
//////////////////////////////////////////////////////////////////////////////
#pragma package(smart_init)
#pragma link "DynamicSkinForm"
#pragma link "SkinCtrls"
#pragma link "SkinData"
#pragma link "SkinBoxCtrls"
#pragma link "SkinTabs"
#pragma link "DynamicSkinForm"
#pragma resource "*.dfm"
TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner) {}
//////////////////////////////////////////////////////////////////////////////
//////////////////////Инициализация///////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Form1->Left=244;
  Form1->Top=176;
  Form1->Width=640;
  Form1->Height=333;
  Form1->spSkinData1->LoadFromCompressedFile("MacRemix.skn");
}
//////////////////////////////////////////////////////////////////////////////
void __fastcall TForm1::FORMRES(TObject *Sender, int &NewWidth,
      int &NewHeight, bool &Resize)
{
 Memo1->Width=ceil(Form1->Width/2)-Memo1->Left-11;
 Memo3->Width=Memo1->Width;
 Memo3->Left=Memo1->Width+Memo1->Left+7;
 Form1->spSkinStdLabel3->Left=Memo3->Left;
```

```
 Memo2->Width=Form1->Width-32;
 Memo2->Height=Form1->Height-122;
 //---------------------------------------------------------------------------
 Memo4->Width=ceil(Form1->Width/2)-Memo4->Left-11;
 Memo5->Width=Memo4->Width;
 Memo5->Left=Memo1->Width+Memo4->Left+7;
 Memo4->Height=Form1->Height-122;
 Memo5->Height=Form1->Height-122;
 //---------------------------------------------------------------------------
 Memo1->Height=Memo1->Top+Form1->Height-180;
 Memo3->Height=Memo1->Top+Form1->Height-180;
 spSkinButton1->Top=Memo3->Top+Memo3->Height+20;
 Form1->spSkinEdit2->Top=spSkinButton1->Top;
 Form1->spSkinStdLabel2->Top=spSkinButton1->Top+4;
}
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////Класс "результат"/////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
CLASSVECTORS CLVEC;  //Здесь будет результат умножения
////////////////////////////////////////////////////////////////////////////////
CLASSVECTORS::CLASSVECTORS (int i)
 {
   pole=i;
 };
//////////////////Создаем динамический массив//////////////////////////////////
void CLASSVECTORS::PROCPOLE(int i)
 {
  VECTORS = new float [i];
 };
////////////////////////////////////////////////////////////////////////////////
float CLASSVECTORS::show(int k)
 {
   return (VECTORS[k]);
 };
////////////////////////////////////////////////////////////////////////////////
void CLASSVECTORS::POLE(int k, int SUM)
 {
   VECTORS[k]=SUM;
 };
////////////////////////////////////////////////////////////////////////////////
CLASSVECTORS::~CLASSVECTORS()
 {
  try
   {
    delete[] VECTORS;
   }
  catch (...)
   {}
 }
//////////////////////////Показываем порядок вектора//////////////////////////
int CLASSVECTORS::SHOWPOLE()
 {
  return (pole);
 }
//////////////////Изменение pole/////////////////////////////////////////////////
void CLASSVECTORS::POLES(int i)
 {
   pole=i;
 }
///////////////Перегружаем знак "*"/////////////////////////////////////////////
float* operator *(VECTOR &M1, MATRICA &M2)
 {
     float *mas;
     mas = new float [M1.SHOWPOLE()];
     float SUM;
     for (int i=0;i<M1.SHOWPOLE();i++)
          {
           SUM=0;
           for (int j=0;j<M1.SHOWPOLE();j++)
             {
               SUM=SUM+M1.show(j)*M2.show(i,j);
             };
           mas[i]=SUM;
          };
     return (mas);
 };
////////////////////////////////////////////////////////////////////////////////
////////////////////////Работаем с вектором/////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
VECTOR VEC;   //Создали экземпляр в памяти
//////////////////////Конструктор "по умолчанию"////////////////////////////////
VECTOR::VECTOR(int i)
 {
```

```
   pole=i;
 };
//////////////////////Конструктор копирования//////////////////////////////////
VECTOR::VECTOR(const VECTOR &V)
 {
  randomize;
  vect = new float [VEC.SHOWPOLE()];  //Выделяем память под копию объекта
  for (int i=0;i<VEC.SHOWPOLE();i++)
   {
     vect[i]=V.vect[i];
   };
 }
//////////////////////////////////Деструктор//////////////////////////////////////
VECTOR::~VECTOR()
 {
  try
   {
    delete[] vect;
   }
  catch (...)
   {}
 };
////////////////////Процедура показа вектора//////////////////////////////////
float VECTOR::show(int i)
 {
  return (ceil(vect[i]));   //Округляем
 };
//////////////////////////Показываем порядок вектора//////////////////////////
int VECTOR::SHOWPOLE()
 {
  return (pole);
 }
////////////////////Создаем динамический массив//////////////////////////////////
void VECTOR::PROCPOLE(int i)
 {
  float a,b;
  randomize;
  vect = new float [i];
  for (int ii=0;ii<i;ii++)
      {
        a=random(100);
        a=a/100;
        b=random(100)+a;
        vect[ii]=b;
      };
 };
////////////////////Изменение pole/////////////////////////////////////////////
void VECTOR::POLES(int i)
 {
   pole=i;
 }
////////////////////////////////////////////////////////////////////////////////
//////////////////////////Работаем с матрицей///////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
MATRICA MATR;   //Создали экземпляр в памяти
////////////////////Конструктор "по умолчанию"//////////////////////////////////
MATRICA::MATRICA(int i)
 {
   pole=i;
 };
//////////////////////Конструктор копирования//////////////////////////////////
MATRICA::MATRICA(const MATRICA &M)
 {
  randomize;
  float **matrs;
  matrs = new float *[MATR.SHOWPOLE()]; //matrs - указатель на массив указателей
  for (int i=0;i<MATR.SHOWPOLE();i++)
    matrs[i]= new float [MATR.SHOWPOLE()];//matrs[i] - массив вещественных чисел
  for (int i=0;i<MATR.SHOWPOLE();i++)
   for (int j=0;j<MATR.SHOWPOLE();j++)
      matrs[i][j]=M.matr[i][j];
 }
//////////////////////////////////Деструктор//////////////////////////////////////
MATRICA::~MATRICA()
 {
  try {
  ///////////Очищаем "массив указателей" на "массивы вещественных чисел"////////
  for (int i=0;i<MATRICA::SHOWPOLE();i++)
    delete[] matr[i];
  ////////////////////"Очищаем указатель на массив указателей"//////////////////
  delete[] matr; }
  catch (...)
  {}
```

```
 };
////////////////////Процедура показа матрицы//////////////////////////////////
float MATRICA::show(int i, int j)
 {
  return (ceil(matr[i][j]));   //Округляем
 };
//////////////////////////Показываем порядок матрицы//////////////////////////
int MATRICA::SHOWPOLE()
 {
  return (pole);
 }
////////////////////Создаем динамическую матрицу//////////////////////////////
void MATRICA::PROCPOLE(int i)
 {
  randomize;
  float a,b;
  //----------------------Создаем динамическую матрицу--------------------------
  matr = new float *[i];  //matr - указатель на массив указателей
  for (int ii=0;ii<i;ii++)
     matr[ii]= new float [i];   //matr[ii] - массив вещественных чисел
  //----------------------------------------------------------------------------
  int ii,jj;
  for (ii=0;ii<i;ii++)
    for (jj=0;jj<i;jj++)
      {
       a=random(100);
       a=a/100;
       b=random(100)+a;
       matr[ii][jj]=b;
      };
 };
////////////////////Изменение pole/////////////////////////////////////////////
void MATRICA::POLES(int i)
 {
  pole=i;
 }
////////////////////////////////////////////////////////////////////////////////
//////////////////////Сгенерировать вектор и матрицу////////////////////////////
////////////////////////////////////////////////////////////////////////////////
void __fastcall TForm1::spSkinButton1Click(TObject *Sender)
{
 try
  {
   Form1->spSkinButton1->Enabled=false;
   ////////////////////ГЕНЕРАЦИЯ ВЕКТОРОВ/////////////////////////////////////
   randomize();
   if (spSkinEdit2->Text!="")
         {
         ////////////////////////////////////////////////////////////////////
         if (StrToInt(spSkinEdit2->Text)>0)
           {
             VEC.POLES(StrToInt(spSkinEdit2->Text));
             VEC.PROCPOLE(VEC.SHOWPOLE());
             MATR.POLES(StrToInt(spSkinEdit2->Text));
             MATR.PROCPOLE(MATR.SHOWPOLE());
             CLVEC.POLES(StrToInt(spSkinEdit2->Text));
             CLVEC.PROCPOLE(CLVEC.SHOWPOLE());
           }
         else
           {
             ShowMessage("Число должно быть >0");
             return;
           }
         ////////////////////////////////////////////////////////////////////
         }
   else
      {
       VEC.PROCPOLE(VEC.SHOWPOLE());  //Инициализация по умолчанию
       MATR.PROCPOLE(MATR.SHOWPOLE());
       CLVEC.PROCPOLE(CLVEC.SHOWPOLE());
      };
   ////////////////////////////////////////////////////////////////////////////
   int i=0,j=0;
   Form1->Memo1->Clear();
   for (i;i<VEC.SHOWPOLE();i++)
    {
     Form1->Memo1->Lines->Add(VEC.show(i));
     Application->ProcessMessages();
    };
   AnsiString s;
   Form1->Memo3->Clear();
   for (i=0;i<MATR.SHOWPOLE();i++)
    {
```

```
      s="";
      for (j=0;j<MATR.SHOWPOLE();j++)
       {
         s=s+MATR.show(i,j)+"  ";
         Application->ProcessMessages();
       };
      Form1->Memo3->Lines->Add(s);
    };
   ////////////////////////////////////////////////////////////////////////////
   ////////////////////////Умножение/////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////////
   CLVEC.operator =(operator *(VEC,MATR));
   Memo2->Clear();
   for (int i=0;i<VEC.SHOWPOLE();i++)
     {
       Form1->Memo2->Lines->Add(CLVEC.show(i));  //Вывод в форму
       Application->ProcessMessages();
     };
   ////////////////////////////////////////////////////////////////////////////
   ////////////////////Процедура копирования объектов//////////////////////////
   ////////////////////////////////////////////////////////////////////////////
   Memo4->Clear();
   VECTOR COPYVEC = VEC;
   for (int i=0;i<VEC.SHOWPOLE();i++)
     {
       Form1->Memo4->Lines->Add(COPYVEC.show(i));
       Application->ProcessMessages();
     };
   //---------------------------------------------------------------------------
   Memo5->Clear();
   MATRICA COPYMATR = MATR;  //Конструктор копирования
   for (int i=0;i<MATR.SHOWPOLE();i++)
    {
      s="";
      for (int j=0;j<MATR.SHOWPOLE();j++)
       {
         s=s+MATR.show(i,j)+"  ";
         Application->ProcessMessages();
       };
     Memo5->Lines->Add(s);
    };
   ////////////////////////////////////////////////////////////////////////////
   //////Теперь можно уничтожить динамические переменные в экземплярах класса///
   ////////////////////////////////////////////////////////////////////////////
   VEC.~VECTOR();
   MATR.~MATRICA();
   CLVEC.~CLASSVECTORS();
   VEC.POLES(CONSTT);    //Восстанавливаем в "классе значение по умолчанию"
   MATR.POLES(CONSTT);   //Восстанавливаем в "классе значение по умолчанию"
   CLVEC.POLES(CONSTT);  //Восстанавливаем в "классе значение по умолчанию"
   COPYVEC.~VECTOR();
   COPYMATR.~MATRICA();
   ////////////////////////////////////
   Form1->spSkinButton1->Enabled=true;
  }
 catch (...)
   {
     ShowMessage("Введите число правильно!!!");
     Form1->spSkinEdit2->Text="";
   };
}
////////////////////////////////////END/////////////////////////////////////////
```