Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра сетей и устройств телекоммуникаций

КЛАССЫ И ОБЪЕКТЫ В ЯЗЫКЕ С++

Методические указания
для практических занятий по курсу
«Объектно-ориентированное программирование»
для студентов специальностей 1-45 01 03 «Сети телекоммуникаций»,
1-45 01 05 «Системы распределения мультимедийной информации»,
1-98 98 01 02 «Защита информации в телекоммуникациях»
дневной и заочной форм обучения

Минск БГУИР 2012

УДК 004.43 (076)
ББК 32.973.26-018.1я7
К47

С о с т а в и т е л и:
К. П. Курейчик, В. А. Мельников

Р е ц е н з е н т:
доцент кафедры систем телекоммуникаций учреждения образования
«Белорусский государственный университет
информатики и радиоэлектроники»
кандидат технических наук О. А. Хацкевич

Классы и объекты в языке С++ : метод. указ. для практ. занятий
К47 по курсу «Объектно-ориентированное программирование» для студ. спец. 1-45 01 03 «Сети телекоммуникаций», 1-45 01 05 «Системы распределения мультимедийной информации», 1-98 01 02 «Защита информации в телекоммуникациях» днев. и заоч. форм обуч. / сост. К. П. Курейчик, В. А. Мельников. – Минск : БГУИР, 2012. – 43 с.
ISBN 978-985-488-838-5.

Приводятся основные характеристики классов и объектов в языке программирования С++, а также типовые примеры по данной теме.

УДК 004.43 (076)
ББК 32.973.26-018.1я7

ISBN 978-985-488-838-5

1. КЛАССЫ, ОБЪЕКТЫ И ИХ ОПИСАНИЕ

1.1. Разновидности классов

Классы в объектно-ориентированном проектировании имеют несколько различных форм и используются для различных целей. Перечислим категории, охватывающие большую часть классов:

– управление данными;
– источник данных и посредники в передаче данных;
– классы для просмотра данных;
– вспомогательные или упрощающие проектирование классы.

В ряде случаев классы разбиваются на два класса.

Классы администраторы данных Data Manager (часто называют Data или State) – это классы, основной обязанностью которых является поддержка данных или информации о состоянии чего-либо. Например, для игры в карты основная задача класса Card состоит в том, чтобы хранить масть и ранг карты. Классы администраторы – это фундаментальные блоки проекта, существительные.

Источник данных Data Source – это классы, которые генерируют данные. Посредники при передаче данных (Data Sinks) служат для приема и дальнейшей передачи данных (например запись в файл). В отличие от классов-администраторов источники и посредники не хранят внутри себя данные, а генерируют их по запросу (источники данных) или обрабатывают их при вызове (посредники данных).

Классы просмотра также незаменимы почти в любом приложении. Все программы осуществляют вывод информации (обычно на экран). То есть можно изолировать внутренние данные и методы, осуществляющие их вывод. Полезно отделять собственно объект от его визуализации. Благодаря этому принципу системы, обеспечивающие вывод графической информации, могут быть значительно упрощены. В идеальном случае модель не требует и не содержит информации о своем визуальном представлении. Это позволяет одну и ту же модель применить в разных приложениях.

К вспомогательным классам можно отнести те классы, которые не содержат полезной информации, но облегчают выполнение сложных заданий. Например, связный список карт – колода.

Пример: игра в карты

Класс *Card* может применяться в карточной игре любого типа.

Класс *Card*

Хранит масть и ранг карты
Возвращает цвет карты
Хранит состояние «картинка вверх или вниз»

Рисует карту на экране
Удаляет карту с экрана

Класс *Card* является администратором данных, который хранит и возвращает значения масти и ранга, и рисует карту. Рассмотрим уточнение класса *Card.*

Класс *Card*

Suit – возвращает масть и ранг карты
Color – Возвращает цвет карты
faceUp – проверяет состояние «картинка вверх или вниз»
draw – рисует карту на экране
erase – удаляет карту с экрана

1.2. Инкапсуляция

При программировании с использованием абстрактных типов данных АТД информация скрывается в небольшой части программы. АТД представляет собой совокупность операций, которые определяют поведение абстракций. С одной стороны, программист, который определяет этот АТД, видит значения переменных, которые используются для поддержания внутреннего состояния объекта. Например, для абстрактного типа данных stack пользователь использует описание допустимых операций – *pop*, *push*, *top*. С другой стороны, программисту, реализующему *stack*, необходимо манипулировать конкретными структурами данных. Конкретные детали инкапсулированы в более абстрактный объект.

Каждый экземпляр имеет свою собственную совокупность переменных. Эти значения не должны изменяться клиентами напрямую, а только с помощью методов, ассоциированных с классом.

Объект является комбинацией состояния и поведения. Состояние описывается переменными экземпляра, поведение характеризуется методами. Снаружи клиенты могут узнать только о поведении объектов. Изнутри доступна полная информация о том, как методы обеспечивают необходимое поведение, изменяют состояние и взаимодействуют с остальными объектами.

1.3. Интерфейс и реализация

В терминах объектов:

– объявление класса должно обеспечивать клиента информацией, необходимой для успешной работы, и никакой другой;

– методам должна быть доступна вся информация, необходимая для выполнения их обязанностей, и никакая другая.

Этит принцип, сформулированный Парнасом, делит мир объекта на две части. Имеется внешний образ, наблюдаемый пользователем объекта; и мы будем называть это представление об объекте *интерфейсом* (*interface*). Оно

описывает, как объект взаимодействует с внешним миром. Обратная сторона объекта связана с его реализацией (implementation). Пользователю разрешен доступ только к тому, что описано в интерфейсной части. Реализация определяет, как достигается выполнение обязанностей, заявленных в интерфейсной части.

1.4. Классы и методы в языке C++

Классы C++ предлагают расширение предопределенных системных типов. Класс – это определяемый пользователем тип. Каждый класс представляет уникальное множество объектов и операций над ними (методов). Класс предоставляет средства для создания, манипулирования и уничтожения таких объектов.

В общем случае класс представляется в следующей форме:

class имя класса{список членов};

Описание класса начинается с ключевого слова *class*. Список членов класса определяет собственные элементы класса.

При описании членов классов возможно указание атрибутов управления доступом к элементам классов. Такими атрибутами являются:

– *public:* члены класса доступны извне класса;

– *private:* соответствующие элементы могут использоваться только внутри класса

По умолчанию элементы класса имеют тип *private*. Указанный в описании спецификатор доступа распространяется на все последующие определения, пока не встретится другой спецификатор.

Поскольку пользователя часто интересует открытая часть класса, то в описании она должна идти первой.

Метод, имеющий то же имя, что и класс, и не имеющий возвращаемого значения, называется конструктором.

Метод используется при инициализации создаваемых экземпляров класса.

При программировании с использованием классов часто применяется много методов с небольшим объемом кода, что снижает быстродействие программы. С целью устранения этого недостатка разработан аппарат *inline*-функций (встраиваемые функции). Компилятор преобразует вызов функции в ее код. Функция, определенная (а не просто описанная) в описании класса, считается *inline*. Кроме того, функцию можно определить с ключевым словом *inline* вне протокола класса.

```
inline int card::rank() { return r; }
```

Язык C++ поддерживает неполное описание классов: *class* X. Неполное описание класса позволяет использовать ссылки на эти классы до их полного определения. Объекты типа класс могут использоваться как аргументы функций и как возвращаемые функциями значения.

1.5. Ключевое слово this

Ключевое слово *this* обозначает специальную локальную переменную, доступную в теле любой функции-члена (метода) класса, описанной без спецификации *static*. Переменная *this* не требует описания и всегда содержит указатель соответствующего объекта:

– *this->имя члена* указывает на объект, членом, которого он является;

– **this* представляет собой сам объект и в зависимости от контекста может быть лево- или правосторонней величиной;

– *this* представляет собой адрес объекта.

Пример

```
class C {
int c1,c2;
public:
void init(int b) { c2=b; c1=b+1; }
C & inc() { c1++; c2++; return *this; }
void *adress() { return this; }
void print() { cout << c1 << c2; }
};
void main(void) {
C a;
a.init(10); a.print(); cout << " adress=" << a.adress<< " inc " <<
a.inc().print() << endl;
}
```

Указатель на объект, для которого вызвана функция-член (метод), является её скрытым параметром. На этот неявный параметр можно ссылаться явно как на *this*. В каждой функции класса *x* указатель *this* неявно описан как

```
x* this;
```

и инициализирован так, что он указывает на объект, для которого была вызвана функция-член. *this* не может быть описан явно, это ключевое слово. Класс *x* можно эквивалентным образом описать так:

```
class x {
int m;
public:
int readm() { return this->m; }
};
```

```
class x {
int m;
public:
int readm() { return m; }
};
```

1.6. Функции-члены

Метод в C++ принято называть функцией-членом класса, о пересылке сообщений говорят как о вызове функции-члена класса:

– функции-члены – это функции, которые манипулируют данными-членами класса;

– функции-члены имеют доступ ко всем полям своего класса;

– функции-члены могут быть в закрытой, защищенной и открытой части класса;

– функции-члены могут быть определены внутри или вне (C++) объявления класса. Функции-члены, определенные вне класса, могут быть сделаны *inline*;

– функции-члены могут обращаться к полям или функциям-членам, объявленным после них;

– функции-члены имеют неявно объявленную переменную this;

– функции-члены могут быть static. Такие функции могут непосредственно обращаться и изменять статические поля класса. Статические функции-члены класса не могут быть объявлены const или virtual. К таким функциям можно обращаться через имя класса, а не через имя конкретного экземпляра объекта.

```
– Class Shape {
– static int num;
– Point center;
– public:
– Shape(Point c): center(c) { ++num; }
– ~Shape() { --num; }
– static int Num() { return num; }
– };
– int Shape::num=0;
– void main()
– {
– cout << Shape::Num(); // 0
– Shape s;
– cout << Shape::Num(); // 1
– for(int n=Shape::Num(); n>0; n--) { }
– }
```

Функции-члены могут быть объявлены как *const*, что не позволяет им изменять значение и возвращать неконстантную ссылку или указатель на любое поле класса. Такие функции-члены не могут быть статическими.

```
class Int {
int *v, size, top;
public:
int Pop();
int Top() const; // Не может изменять что-либо
};
void f(const int &s) {
int a =s.Top(); // Правильно: Top() константный член
```

```
class A {
int a;
public:
A(){a=10; }
Int & get() const; // Ошибка возвращает ссылку
};
int & A::get() const { return a; }
int main(int argc, char*
```

```
    int b=s.Pop(); // Ошибка : s не
может быть модифицирован
    }
```

```
argv[]){
    A b;
    cout<< " " << b.get();
    return 0;
    }
```

1.7. Данные-члены и управление доступом к элементам классов

Данные-члены – это набор взаимосвязанной информации, возможно, различных типов, объединенной в один объект.

Данные-члены могут находиться в закрытой (private), защищенной (protected) или открытой (public) части класса.

Данные-члены могут иметь статический класс памяти (static). Поля, имеющие статический класс памяти, совместно используются всеми объектами класса. К ним возможен доступ через имя класса (с использованием операции разрешения доступа), а не через контекстный объект класса. Статические поля могут быть инициализированы; если нет, то инициализируется значением ноль.

Данные-члены могут быть объявлены как const. Константные данные должны быть инициализированы в каждом определении конструктора. Имена полей и начальные значения заключаются в скобки, отделяются от списка аргументов конструктора двоеточием.

Данные-члены могут быть переменными другого класса. В этом случае требуется явная инициализация, если только поля не имеют конструктора по умолчанию.

1.8. Создание и инициализация объектов

Наиболее важным в данном случае является вопрос о выделении и освобождении памяти как крайне дефицитного системного ресурса.

Прежде всего это стек и куча. Вопрос о выделении памяти через стек или кучу связан с тем, как выделяется и освобождается память. Разделяют автоматические и динамические переменные. Для автоматических переменных память создается при входе в процедуру или блок, которые управляют этими переменными. При выходе из блока память (автоматически) освобождается. Многие языки используют термин статический (static) для обозначения переменных, автоматически размещаемых в стеке (в данном случае речь не идет о переменных типа static в C++). Для динамического выделения памяти используется оператор new. Память выделяется по специальному запросу со стороны пользователя.

Автоматические переменные уничтожаются автоматически, динамические – по запросу со стороны пользователя ключевым словом delete в C++.

1.9. Создание и инициализация в C++. Конструкторы

Использование для обеспечения инициализации объекта класса функций вроде set_date() (установить дату) неэлегантно и чревато ошибками. Поскольку нигде не утверждается, что объект должен быть инициализирован, то программист может забыть это сделать, или (что приводит, как правило, к столь же разрушительным последствиям) сделать это дважды. Есть иной подход: дать возможность программисту описать функцию, явно предназначенную для инициализации объектов. Поскольку такая функция конструирует значения данного типа, она называется конструктором. Конструктор распознается по тому, что имеет то же имя, что и сам класс. Например:

```
class date {
date(int, int, int);
};
```

Метод конструктор автоматически и неявно вызывается каждый раз, когда создается объект, принадлежащий к соответствующему классу. Это происходит или во время объявления переменной или, когда объект создается динамически с помощью оператора new или когда применяются временные переменные.

Иногда необходимо обеспечить несколько способов инициализации объекта класса. Это можно сделать, задав несколько конструкторов. *Например:*

```
class date {
int month, day, year;
public:
date(int, int, int); // день месяц год
date(char*); // дата в строковом представлении
date(int); // день, месяц и год сегодняшние
date(); // дата по умолчанию: сегодня
};
data::data(){
month=01, day=22, year=1999;
}
date today(4);
date july4("Июль 4, 1983");
date now; // инициализируется по умолчанию
```

Тело конструктора часто представляет собой последовательность операторов присваивания. Они могут быть заменены инициализаторами в заголовке функции.

```
Data(int x,int y,int z):month(x),day(y),year(z) {}
```

Динамическое создание переменной выглядит следующим образом:

```
Data *d, *g;
D=new data(1,3,1987);
G=new data;
```

Конструкторы подчиняются тем же правилам относительно типов параметров, что и перегруженные функции. Если конструкторы существенно

различаются по типам своих параметров, то компилятор при каждом использовании может выбрать правильный.

Размножение конструкторов в примере с *date* типично. При разработке класса всегда есть соблазн обеспечить все, поскольку кажется проще обеспечить какое-нибудь средство просто на случай, что оно кому-то понадобится, или потому, что оно изящно выглядит, чем решить, что же нужно на самом деле. Последнее требует больших размышлений, но обычно приводит к программам, которые меньше по размеру и более понятны. Один из способов сократить число родственных функций – использовать параметры со значением по умолчанию. В случае *date* для каждого параметра можно задать значение по умолчанию, интерпретируемое как «по умолчанию принимать: today» (сегодня).

```
class date {
int month, day, year;
public:
date(int d =0, int m =0, int y =0);
date(char*); // дата в строковом представлении
};
```

1.10 Констукторы и массивы объектов

Чтобы описать вектор объектов класса, имеющего конструктор, этот класс должен иметь конструктор, который может вызываться без списка параметров. Нельзя использовать даже параметры по умолчанию. Например:

```
table tblvec[10];
```

будет ошибкой, если нет конструктора без параметров, так как для *table::table*(*int sz*=10) требуется целый параметр. Нет способа задать параметры конструктора в описании вектора. Чтобы можно было описывать вектор таблиц *table*, можно модифицировать описание *table*, например, так:

```
class table {
void init(int sz); // как старый конструктор
public:
table(int sz) { init(sz); } // как раньше, но без по умолчанию
table() { init(10); } // по умолчанию
};
```

Когда вектор уничтожается, деструктор должен вызываться для каждого элемента этого вектора. Для векторов, которые не были размещены с помощью new, это делается неявно. Однако для векторов в свободной памяти это не может быть сделано неявно, поскольку компилятор не может отличить указатель на один объект от указателя на первый элемент вектора объектов. Например:

```
void f() {
table* t1 = new table;
table* t2 = new table[10];
table* t3 = new table[10];
```

```
delete t1; // одна таблица
delete t2; // неприятность: 10 таблиц
delete[] t3;
}
```

Компилятор не может найти число элементов вектора из объема выделенной памяти, потому что распределитель свободной памяти не является частью языка и может быть задан программистом.

Конструкторы могут инициализировать массивы объектов класса таким образом, как и массивы встроенных типов. Максимальное число элементов может быть опущенно, если оно равно числу инициализирующих значений. Если максимальное число элементов больше числа значений, то оставшиеся значения инициализируются при помощи конструктора по умолчанию. Если это не конструктор по умолчанию, то все же значения должны быть указаны. Если заданы все значения для данного размера массива, то фигурные скобки при указании списка значений могут быть опущены.

```
class Phone {
int a,b,c;
public:
Phone(int a1,int b1,int c1):a(a1),b(b1),c(c1) {}
};
Phone office[]= { // Компилятор вычислит размер за нас
900, 800, 905,
678,456,546 };
Phone office[3]= { // Требуется конструктор по умолчанию, которого у
Phone нет
890, 790,343,
238, 279, 564
};
```

1.11. Деструктор

Определяемый пользователем тип имеет конструктор, который обеспечивает требуемую инициализацию переменных, используемых объектом. Для многих типов также требуется обратное действие – деструктор – чтобы обеспечить соответствующую очистку объектов этого типа. Имя деструктора для класса *X* есть *~X*() (дополнение конструктора). В частности, многие типы используют некоторый объем памяти из свободной памяти, который выделяется конструктором и освобождается деструктором. Вот, например, традиционный стековый тип, из которого для краткости полностью выброшена обработка ошибок:

```
class char_stack {
int size;
char* top, * s;
public:
char_stack(int sz) { top=s=new char[size=sz]; }
```

~char_stack() { delete []s; } // деструктор
*void push(char c) { *(top++) = c; }*
*char pop() { return *(--top);}*
};

Когда char_stack выходит из области видимости, вызывается деструктор:

void f(){
char_stack s1(100), s2(200);
s1.push('a'); s2.push(s1.pop());
char ch = s2.pop();
cout << char(ch) << "\n";
}

Когда вызывается *f()*, конструктор *char_stack* вызывается для *s1*, чтобы выделить вектор из 100 символов, и для *s2*, чтобы выделить вектор из 200 символов. При возврате из *f()* эти два вектора будут освобождены.

1.12. Конструктор копирования

Объект класса без конструкторов можно инициализировать путем присваивания ему другого объекта этого класса. Это можно делать и тогда, когда конструкторы описаны. Например:

date d = today; // инициализация посредством присваивания

По существу имеется конструктор по умолчанию, определенный как почленная (в ранних версиях C++ побитовая) копия объекта того же класса. Если для класса *X* такой конструктор по умолчанию нежелателен, его можно переопределить конструктором с именем *X(X&).*

Семантика вызова по значению требует, чтобы локальная копия типа параметра создавалась и инициализировалась от значения выражения, переданного как фактический параметр. Для этого необходим *конструктор копии.* Компилятор предоставляет такой конструктор по умолчанию. Его сигнатура

stack::stack(const stack&);

Компилятор производит копирование путем почленной инициализации. Это не всегда применимо. В большинстве случаев указатель является адресом объекта, удаляемого при выходе из контекста. Однако код, в котором производится действие по дублированию значения указателя, а не объекта, на который он указывает, может быть ошибочным. Дело в том, что удаление воздействует на другие экземпляры, все еще предполагающие, что объект существует. Поэтому важно, чтобы класс имел свою собственную явно определенную копию конструктора.

Stack::stack(const stack& str) {
s=new char [str.max_len]; max_len=str.max__len; top=str.top;
memcpy(s,str.s,max_len);
}

Конструктор копирования вызывается при возвращении значения объекта из функции

stack f(stack r) // вызов конструктора копирования при создании копии

{

return r; // вызов конструктора копирования для создании копии при возвращении

}

Конструктор копирования вызывается в случае инициализации при объявлении:

Stack s(10); // конструктор с одним параметром

Stack d=s; // конструктор копирования

Все хорошо разработанные классы должны иметь конструктор копирования, особенно те, которые имеют дело с динамической памятью, открытыми файлами, любыми указателями.

1.13. Резюме

В C++ существуют специальные функции-члены класса, которые определяют, как объекты класса создаются, инициализируются, копируются и уничтожаются. Важнейшие из них – конструкторы и деструкторы. Им свойственны многие черты обычных функций-членов класса, но существуют и свои особенности:

– конструкторы и деструкторы не могут описываться как функции, возвращающие значение (даже типа void);

– конструкторы могут иметь аргументы. В качестве параметров могут быть элементы, получающие значения по умолчанию. Деструктор не имеет параметров;

– имя конструктора совпадает с именем класса. Имя деструктора – имя класса, которому предшествует символ ~;

– нельзя получить адрес конструктора или деструктора;

– если они явно не описаны, то конструкторы и деструкторы автоматически создаются компилятором;

– конструктор не может быть вызван как обычная функция. Деструктор может быть вызван как обычная функция с указанием полного имени:

*Int s(100), *ps=new Int(50);*

s.~Int(); // явное разрушение s

*this->~Int(); // Явное разрушение *this (may be error this=ps ?)*

– компилятор автоматически вставляет вызовы конструктора и деструктора при описании и уничтожении объекта.

– конструкторы могут быть и в закрытой, и в защищенной, и в открытой части класса. Если конструктор находится в закрытой части класса, то объект этого класса с использованием такого конструктора могут создавать только те, кто имеет доступ к закрытой части;

– деструктор автоматически вызывается компилятором:

при выходе из области видимости;

при создании временных объектов для преобразования в выражениях или для передачи функциям аргументов;

когда возвращенные функцией значения более не нужны;

при выполнении операции delete для объектов, размещенных в динамической памяти;

– конструктор не может быть ни const, ни volatile, ни static, ни virtual;

– конструктор может быть вызван тремя эквивалентными способами (пользователю предоставляется право выбора):

Int good(100);
Int bad=Int(100);
Int ff=100;

– конструкторы используются для инициализации константных полей и полей, которые являются переменными класса. Константное поле должно быть инициализировано в каждом определении конструктора. Классы с константными полями должны иметь по крайне мере один конструктор:

```
class Int {
const int size;
public:
Int(int i): size(i) { }
};
```

Если у класса есть конструктор, то он вызывается всегда, когда создается объект класса. Если у класса есть деструктор, то он вызывается всегда, когда объект класса уничтожается. Объекты могут создаваться как:

– автоматический объект: создается каждый раз, когда его описание встречается при выполнении программы, и уничтожается каждый раз при выходе из блока, в котором оно появилось;

– статический объект: создается один раз, при запуске программы, и уничтожается один раз, при ее завершении;

– объект в свободной памяти: создается с помощью операции *new* и уничтожается с помощью операции *delete*;

– объект-член: как объект другого класса или как элемент вектора.

1.14. Пример класса Строка

```
#include <string.h>
class StringHolder {
char *contens;
public:
StringHolder(char *aString=0);
~StringHolder(void);
char *getContens(void);
void setContens(char *aString);
};
StringHolder::StringHolder(char *aString) {
if(aString) { contens=new char[strlen(aString)+1]; strcpy(contens,aString); }
```

```
else {contens=new char[1]; *contens='\0'; }
}
StringHolder::~StringHolder(void) { delete contens; }
char *StringHolder::getContens(void) { return contens; }
void StringHolder::setContens(char *aString) {
delete contens;
contens=new char[strlen(aString)+1]; strcpy(contens,aString);
}
void main()
{
StringHolder str1,str2("Object2"), *str3, *str4;
str3=new StringHolder("object3");
str4=new StringHolder;
cout << str1.getContens(); cout << str2.getContens();
cout << str3->getContens(); cout << str4->getContens();
}
```

2. ПРАКТИЧЕСКИЙ ПРИМЕР СОЗДАНИЯ КЛАССА

Создадим классс, обеспечивающий отображение стрелочного измерительного прибора, например вольтметра. Среда подготовки *MS Visual* C++ 6.0.

```
//h-file
#include "stdafx.h"
#include "math.h"

// settings  parametrs of the DMeter
class CMeterSettings
{
public:
        CMeterSettings();
        virtual      ~CMeterSettings();
        afx_msg void SetFont(CFont* pFont);         //set font
        afx_msg void SetColorBkText(COLORREF color);//set bk color
        afx_msg void SetColorText(COLORREF color);//set text color
        CFont* m_pFont,
                m_Font;   //font on default
        //section color
        COLORREF m_colorBk,//color BkText
                        m_colorText;//colorText
};

class CMeter : public CStatic
{
```

```
public:
    CMeter();
    virtual     ~CMeter();
    CMeterSettings SetMetTop,
                   SetMetTitle,
                       SetMetScale,
                       SetMetData;
public:
    afx_msg BOOL Connect(UINT nID,CWnd *pWnd);//attach
    afx_msg void GetDataMeter(double dData,BOOL bFlag); //current data
    afx_msg void GetMaxData(double dData,BOOL bFlag);// mas value of a
Data
    afx_msg void SetGrades(int nGrad,int nGradMiddle);//set value of
grades for first and second sectore
    afx_msg void SetColorBk(COLORREF colorBk);//color of Bk
    afx_msg void SetColorLineIndicator(COLORREF color);//color line
indicator
    afx_msg void SetTextTitle(CString pStrTitle);//text of title
    afx_msg void SetTextTopTitle(CString pStrTopTitle);//text of toptitle
    afx_msg void SetTextData(CString pStrData);//text of data
    //section SetColorLine
    afx_msg void SetColorLowLine(COLORREF color);
    afx_msg void SetColorMiddleLine(COLORREF color);
    afx_msg void SetColorHightLine(COLORREF color);
    afx_msg void SetColorFontLine(COLORREF color);
    afx_msg void OnPaint();

protected:
    //common
    CRect afx_msg MyRect(CRect rect);
    int afx_msg SetCoordinates(int nGrad,int nRadius,CString
pStrFunc);//coordinates
    void afx_msg DrawScale(CDC* pDC,int nStartX,int nStarY, int
nEndX,int nEndY); //scale of the meter
    CRect afx_msg DrawRect3D(CDC* dcMem,CRect rc); //draw bound of
the meter
    void afx_msg SetTextMeter(CDC* dcMem,CRect rc,BOOL bTop);//set
text- title and data
    void afx_msg DrawLineMeter(CDC* dcMem,CRect rc,int
nRadius);//draw linemeter
    CRect afx_msg SetRectLR(CDC* dcMem,CRect rect,CPoint
point,CString pStr,COLORREF crColor);
    void afx_msg DrawLineIndicator(CDC* dcMem,CRect rect,CBrush*
brush,int nI);
```

```
        void  afx_msg DrawFillScale(CDC* pDC,CRect rect,int nRadius);//draw
Fill Scale
        void  afx_msg DrawMXScale(CDC*dcMem, CRect NewRect,CRect rect,
int nRadius);//draw MX Scale
        //section color-brush
        CBrush pBrush,
               pBrushLineIndicator,
               pBrushFrame,//colorBk
           pBrushEllipse;//color of ellipse
        //section color-pen
        CPen   pPenLow,//Pen for dMeterLine        of low part
               pPenMiddle,        //pen of the part middle
               pPenHight,// pen of the part hight
               pPenFrame,// pen of the left and top frame
               pPenFrameShadow,// shadow pen
               pPenLineFont, //pen of the meter scale
               pPenEllipse;//pen of the ellipse
        //section scale
        double m_dMaxScale,//scale of max value
               m_dCurrentScale,//scale of current value
               m_dLimitAngleRad;//limin angle
        //section grades
        int m_nGrad,
               m_nGradMiddle,
               m_nGradEnd;

        //section text
        CString m_pStrTitle,
               m_pStrTopTitle,
               m_pStrData,
               m_pStrMax;

    COLORREF m_colorText; //color
DECLARE_MESSAGE_MAP()
};

//cpp
#include "stdafx.h"
#include "CMeter.h"

//*****************class CMeterSettings*****************
CMeterSettings::CMeterSettings()
{
        m_Font.CreateStockObject(SYSTEM_FONT);
```

```
        m_pFont=&m_Font;
        m_colorBk=GetSysColor(COLOR_WINDOW);
        m_colorText=GetSysColor(COLOR_WINDOWTEXT);
}

CMeterSettings::~CMeterSettings()
{
}

// Function name : CMeterSettings::SetFont
// Description   : set font
// Return        : void
// Argument      : CFont* pFont - point on the CFont
void CMeterSettings::SetFont(CFont* pFont)
{
        m_pFont=pFont;
}

// Function name : CMeterSettings::SetColorBkText
// Description   : set color
// Return        : void
// Argument      : COLORREF color - color of the BkText
void CMeterSettings::SetColorBkText(COLORREF color)
{
        m_colorBk=color;
}

// Function name : CMeterSettings::SetColorText
// Description   : set color
// Return        : void
// Argument      : COLORREF color - color of the Text
void CMeterSettings::SetColorText(COLORREF color)
{
   m_colorText=color;
}

//******************end class CMeterSettings

//******************class  CMeter*******************

BEGIN_MESSAGE_MAP(CMeter, CStatic)
ON_WM_PAINT()  //event on_wm_paint
END_MESSAGE_MAP()
```

```
CMeter::CMeter()
{
        //section brush
        pBrush.CreateSolidBrush(RGB(50,100,255));
        pBrushFrame.CreateSolidBrush(RGB(100,255,255));
        pBrushEllipse.CreateSolidBrush(RGB(200,150,200));
        pBrushLineIndicator.CreateSolidBrush(RGB(0,0,0));
        //section pen frame

    pPenFrame.CreatePen(PS_SOLID,2,GetSysColor(COLOR_3DHIGHLIGHT));
        pPenFrameShadow.CreatePen(PS_SOLID,2,GetSysColor(COLOR_3DS
HADOW));

        pPenLineFont.CreatePen(PS_SOLID, 2, RGB(255,255,0));
        pPenEllipse.CreatePen(PS_SOLID,1,RGB(0,0,255));
        //section value and notefication
        m_dMaxScale=1;m_dCurrentScale=0;
        m_nGrad=60;
        m_pStrTitle=m_pStrData="?????";

}

CMeter::~CMeter()
{
}

// Function name :CMeter::GetMaxData
// Description   : get max value of a data
// Return        : void
// Argument      : double dData - max value of a dData
void CMeter::GetMaxData(double dData,BOOL bFlag)
{
        m_dMaxScale=dData;
        bFlag==TRUE ? m_pStrMax.Format("%.1f",dData):
        m_pStrMax.Format("%d",(int)dData);

        RedrawWindow();
}

// Function name : CMeter::GetDataMeter
// Description   : get current data for CMeter
// Return        : void
// Argument      : double dData - current a data for CMeter
void CMeter::GetDataMeter(double dData,BOOL bFlag)
```

```
{
        m_dCurrentScale=dData;
        bFlag==TRUE ? m_pStrData.Format("%.1f",dData):
        m_pStrData.Format("%d",(int)dData);
//      m_pStrData.Format("%.1f",dData);
        RedrawWindow();
}


// Function name : CMeter::SetTextData
// Description   : set text (data);
// Return        : void
// Argument      : CString pStrData - text
void CMeter::SetTextData(CString pStrData)
{
        m_pStrData=pStrData;
        RedrawWindow();
}


// Function name : CMeter::SetTextTitle
// Description   : set text (title);
// Return        : void
// Argument      : CString pStrTitle text
void CMeter::SetTextTitle(CString pStrTitle)
{
        m_pStrTitle=pStrTitle;
   RedrawWindow();
}

void CMeter::SetTextTopTitle(CString pStrTopTitle)
{
        m_pStrTopTitle=pStrTopTitle;
}

// Function name : CMeter::SetColorBk
// Description   : set color Bk
// Return        : void
// Argument      : COLORREF colorbk - color of the Bk
void CMeter::SetColorBk(COLORREF colorBk)
{
        pBrush.DeleteObject();
        pBrush.CreateSolidBrush(colorBk);
}
```

```
// Function name : CMeter:: SetColorLowLine
// Description   : set color  begin of the lineMeter
// Return        : void
// Argument      : COLORREF color - color of the lowline
void CMeter::SetColorLowLine(COLORREF color)
{
        pPenLow.DeleteObject();
        pPenLow.CreatePen(PS_SOLID, 4, color);
        RedrawWindow();
}

// Function name : CMeter:: SetColorMiddleLine
// Description   : set color  begin of the lineMeter
// Return        : void
// Argument      : COLORREF color - color of the lowline
void CMeter::SetColorMiddleLine(COLORREF color)
{
        pPenMiddle.DeleteObject();
        pPenMiddle.CreatePen(PS_SOLID,4,color);
        RedrawWindow();
}

// Function name : CMeter:: SetColorHightLine
// Description   : set color  begin of the lineMeter
// Return        : void
// Argument      : COLORREF color - color of the lowline
void CMeter::SetColorHightLine(COLORREF color)
{
        pPenHight.DeleteObject();
        pPenHight.CreatePen(PS_SOLID,4,color);
        RedrawWindow();
}

// Function name : CMeter:: SetColorFontLine
// Description   : set color  fontline of the lineMeter
// Return        : void
// Argument      : COLORREF color - color of the lowline
void CMeter::SetColorFontLine(COLORREF color)
{
        pPenLineFont.DeleteObject();
        pPenLineFont.CreatePen(PS_SOLID,2,color);
        RedrawWindow();
```

```
}

void CMeter::SetColorLineIndicator(COLORREF color)
{
	pBrushLineIndicator.DeleteObject();
	pBrushLineIndicator.CreateSolidBrush(color);
	RedrawWindow();
}

// Function name CMeter: Connect
// Description	: Set value of object
// Return	:  void
// Argument	: INT nID - IDC_ of object Static
// Argument	: CWnd *pWnd - point of parent class - is this
BOOL CMeter::Connect(UINT nID,CWnd *pWnd)
{
	if(!SubclassDlgItem(nID,pWnd))	return FALSE;;
	CRect rect;
	CMeter::GetClientRect(&rect);
	int nH=rect.Height()/40;
	pPenLow.CreatePen(PS_SOLID,nH, RGB(255,255,0));//jelow
	pPenMiddle.CreatePen(PS_SOLID,nH, RGB(0,255,0));
	pPenHight.CreatePen(PS_SOLID,nH, RGB(255,0,0));
	return TRUE;
}

// Function name :: CMeter:MyRect
// Description   :set rect
// return	: CRect rect
// Argument	: CRect rect - rect of the ClientArea
CRect CMeter::MyRect(CRect rect)
{
	CRect rc;
	if (rect.Height()>=rect.Width())
	{
		rc.top=rect.top;
		rc.bottom=rect.right;
		rc.left=rect.left;
		rc.right=rect.right;
	} else
	{
		rc.top=rect.top;
		rc.left=rect.left;
		rc.bottom=rect.bottom;
```

```
                rc.right=rect.bottom;
            }
        return rc;
    }

    // Function name:CMeter::SetCoordinates
    // Description  : set coordinates of radius-vector
    // Return       : int
    // Argument     : int Grad - grade of the
    // Argument     : int nRadius  - radius-vector
    // Argument     : CString pStrFunc - name of the functio Cos of Sin
    int CMeter::SetCoordinates(int nGrad,int nRadius,CString pStrFunc)
    {
            int nCoordinates;
            nCoordinates=pStrFunc=="Cos" ?
(int)(nRadius*cos(3.14*nGrad/180)):
            (int)(nRadius*sin(3.1416*nGrad/180));
            return nCoordinates;
    }

    // Function name : CMeter::DrawScale
    // Description   : drawscale of the Meter
    // Return        : void
    // Argument      : CDC* pDC - client aria
    // Argument      : int nStartX,,int nStartY,int nEndX,int nEndY - data
    void  CMeter::DrawScale(CDC  *pDC,int  nStartX,int  nStartY,int  nEndX,int
nEndY)
    {
      pDC->MoveTo(nStartX,nStartY);
      pDC->LineTo(nEndX,nEndY);
    }

    // Function name : CMeter : SetGrades
    // Description   : set grad of part dmeter
    // Return        : void
    // Argument      : int nGrad - of the firs part
    // Argument      : int nGradMiddle - of the second part
    void CMeter::SetGrades(int nGrad,int nGradMiddle)
    {
       m_nGrad=(nGrad<0 || nGrad>180) ? 60:nGrad;
            m_nGradMiddle=(nGradMiddle<0 || nGradMiddle>180) ?
60:nGradMiddle;
            m_nGradEnd=180-(m_nGradMiddle);
            m_nGradEnd=m_nGradEnd<0 ? 0:m_nGradEnd;
```

```
}

// Function name:CMeter::SetTextMeter
// Description  : set text of Title and Data
// Return       : void
// Argument     : CDC* dcMem device context
// Argument     : CRect rc - client aria
// Argumen      : BOOL bTop -  text lie ofn the top or bottom
void CMeter::SetTextMeter(CDC* dcMem,CRect rc,BOOL bTop)
{
        bTop==TRUE ? dcMem->DrawText(m_pStrTitle,rc,DT_CENTER):
        dcMem->DrawText(m_pStrData,rc,DT_CENTER);
}


// Function name:CMeter::SetRectLR
// Description  :get rect and set MXText
// Return       :CRect rect
// Argument     :CDC* dcMem device context
// Argument     :CPoint point - coordinates of lef and top of begin rect for text
// Argument     :CString pStr - MXText
CRect CMeter::SetRectLR(CDC*dcMem,CRect rect,CPoint point,CString
pStr,COLORREF crColor)//get rect and set MXText
{
        CRect rc;
        TEXTMETRIC tm;
        dcMem->GetTextMetrics(&tm);
        int nHeight=tm.tmHeight+tm.tmExternalLeading;//height
        rc.left=point.x;
        rc.top=point.y;
        rc.right=rect.right-1;
        rc.bottom=rc.top+nHeight;
        dcMem->SetTextColor(RGB(255,255,255));
        dcMem->SetBkColor(crColor);
        dcMem->DrawText(pStr,rc,DT_CENTER);

        int OldMode=dcMem->SetBkMode(TRANSPARENT);
        rc.left +=1; rc.top +=1;
        rc.right +=1;rc.bottom +=1;
        dcMem->SetTextColor(m_colorText);
    dcMem->DrawText(pStr,rc,DT_CENTER);
        dcMem->SetBkMode(OldMode);
        return rc;
}
```

```
// Function name : CMeter:DrawRect
// Description   : draw bound of the Rectangle
// Return        : void
// Argument      : CDC* dcMem - context device
// Argument      : CRect rc - client aria
CRect CMeter::DrawRect3D(CDC* dcMem,CRect rc)
{       //section of hightlight
        dcMem->MoveTo(rc.left,rc.bottom);
        dcMem->LineTo(rc.left,rc.top);
        dcMem->LineTo(rc.right,rc.top);
        dcMem->MoveTo(rc.left+1,rc.bottom-1);
        dcMem->LineTo(rc.left+1,rc.top+1);
        dcMem->LineTo(rc.right-1,rc.top+1);
        int ndelta=rc.Width()/6;
    int y=rc.bottom-(int)(rc.bottom/4.2);
        //section of horizontal bottom of spliter
        dcMem->MoveTo(rc.left+ndelta,y+3);
        dcMem->LineTo(rc.right-ndelta,y+3);
        //section of shadow
        dcMem->SelectObject(&pPenFrameShadow);
        dcMem->MoveTo(rc.right-1,rc.top+1);
        dcMem->LineTo(rc.right,rc.bottom);
        dcMem->LineTo(rc.left,rc.bottom);
        dcMem->MoveTo(rc.left,rc.bottom-1);
        dcMem->LineTo(rc.right-1,rc.bottom-1);
        dcMem->LineTo(rc.right-1,rc.top+1);
        //section of horizontal top        of spliter
        dcMem->MoveTo(rc.left+ndelta,y);
        dcMem->LineTo(rc.right-ndelta,y);
        CRect rect;
        rect.left=rc.left+ndelta; rect.right=rc.right-ndelta;
        rect.top=y+1;rect.bottom=y+3;
        return rect;
}


// Function name:CMeter::DrawLineMeter
// Description  : draw line        of the Pie
// Return       : void
// Argument     : CDC* dcMem - point on the context device
// Argument     : CRect rc - client aria
void CMeter::DrawLineMeter(CDC* dcMem,CRect rc, int nRadius)
{
```

```
            m_dCurrentScale=m_dCurrentScale>m_dMaxScale ?
m_dMaxScale:m_dCurrentScale;
        rc.left +=5;rc.right -=5;rc.top +=5;rc.bottom -=5;
            double dKGrad=m_dMaxScale/180;  //relative value
            int dGrad=(int)(m_dCurrentScale/dKGrad); //number of grad
            int nW=rc.Width()/14;
            dGrad -=nW;
        int x1=rc.left; int y1=rc.top;
            int x2=rc.right; int y2=rc.bottom;
            int x3,y3;
            x3=rc.left+(nRadius-SetCoordinates(dGrad,nRadius,"Cos"));
            y3=rc.top+(nRadius-SetCoordinates(dGrad,nRadius,"Sin"));
            CPoint start,stop;  start.x=x3;start.y=y3;
            nW=rc.Width()/10;
            stop.x=rc.left+(nRadius-
SetCoordinates(dGrad+nW,nRadius,"Cos"));;//rc.CenterPoint().x;
            stop.y=rc.top+(nRadius-
SetCoordinates(dGrad+nW,nRadius,"Sin"));;//rc.CenterPoint().y;
            dcMem->Pie(rc,stop,start);
    }

    // Function name:CMeter::DrawLineIndicator
    // Description  :draw line indicator - IntoSystem
    // Return       : void
    // Argument     : CDC* dcMem- point on the device context
    // Argument     : CRect rect - clien area
    // Argument     : CBrush *brush - point of the CBrush
    // Argument     : int nI - step of the scale 0-180
    void CMeter::DrawLineIndicator(CDC* dcMem,CRect rect,CBrush*brush,int
nI)
    {
        double dKGrad=(double)rect.Width()/m_dMaxScale;
        BOOL bFlag=FALSE;
        double dCurrent=m_dCurrentScale;
        if (m_dCurrentScale>m_dMaxScale)
        {
            bFlag=TRUE;
            dCurrent=m_dMaxScale;
        }
        int dGrad=(int)(dCurrent*dKGrad);//max dGrad=rect.Width()
        rect.right=rect.left+dGrad;
        if (bFlag==TRUE)
        {
         CBrush brushRed;
```

```
            brushRed.CreateSolidBrush(RGB(255,0,0));
            dcMem->FillRect(rect,&brushRed);
           }
           else
           dcMem->FillRect(rect,brush);
    }

    void CMeter::DrawFillScale(CDC *dcMem,CRect NewRect,int nRadius)
    {
           for (int i=0;i<181;i++)
           {
                  int x3=NewRect.left+(nRadius-SetCoordinates(i,nRadius,"Cos"));
                  int y3=NewRect.top+(nRadius-SetCoordinates(i,nRadius,"Sin"));
                  int x4=NewRect.left+(nRadius-SetCoordinates(i,nRadius,"Cos"));
                  int y4=NewRect.top+(nRadius-SetCoordinates(i,nRadius,"Sin"));
                  dcMem->SelectObject(&pPenLineFont);

                  if (i==0)
                  {
                         int deltaX=SetCoordinates(0,(int)nRadius/8,"Cos");
                         int deltaY=SetCoordinates(0,(int)nRadius/8,"Sin");
                         DrawScale(dcMem,x4,y4,x4-deltaX,y4-deltaY);
                  }

                  if (i==m_nGrad-1 && i<=90)
                  {
                         int          x3=NewRect.left+(nRadius-SetCoordinates(i-
1,nRadius,"Cos"));
                         int          y3=NewRect.top+(nRadius-SetCoordinates(i-
1,nRadius,"Sin"));
                         int
    deltaX=SetCoordinates(m_nGrad,(int)nRadius/8,"Cos");
                         int
    deltaY=SetCoordinates(m_nGrad,(int)nRadius/8,"Sin");
                         DrawScale(dcMem,x3,y3,x3-deltaX,y3-deltaY);
                  }

                  if (i==m_nGrad-2 && i>90)
                  {
                         int     deltaX=SetCoordinates(m_nGrad-
2,(int)nRadius/8,"Cos");
                         int     deltaY=SetCoordinates(m_nGrad-
2,(int)nRadius/8,"Sin");
                         DrawScale(dcMem,x3,y3,x3+deltaX+6,y3-deltaY);
```

```
                }
                if (i==m_nGradMiddle && i<=90)
                {
                        int
        deltaX=SetCoordinates(m_nGradEnd,(int)nRadius/8,"Cos");
                        int
        deltaY=SetCoordinates(m_nGradEnd,(int)nRadius/8,"Sin");
                        DrawScale(dcMem,x3,y3,x3+deltaX+2,y3-deltaY);
                }
                if (i==m_nGradMiddle+1 && i>90)
                {
                        int
        deltaX=SetCoordinates(m_nGradEnd,(int)nRadius/8,"Cos");
                        int
        deltaY=SetCoordinates(m_nGradEnd,(int)nRadius/8,"Sin");
                        DrawScale(dcMem,x3,y3,x3+deltaX+1,y3-deltaY);
                }
                  if (i==180)
                {
                        int     deltaX=SetCoordinates(180,(int)nRadius/8,"Cos");
                        int     deltaY=SetCoordinates(180,(int)nRadius/8,"Sin");
                        DrawScale(dcMem,x3,y3,x3-deltaX,y3-deltaY+1);
                }
        }
        }

        void CMeter::DrawMXScale(CDC* dcMem,CRect NewRect,CRect rect,int
nRadius)
        {
                int x1=NewRect.left-2;
                int y1=NewRect.top-2;
                int x2=NewRect.right-2;
                int y2=NewRect.bottom-2;
                for (int i=0;i<178;i++)
                {
                        int                             x3=NewRect.left-2+(nRadius-
SetCoordinates(i+3,nRadius,"Cos"));
                        int                             y3=NewRect.top-2+(nRadius-
SetCoordinates(i+3,nRadius,"Sin"));
                        int                             x4=NewRect.left-2+(nRadius-
SetCoordinates(i,nRadius,"Cos"));
                        int                             y4=NewRect.top-2+(nRadius-
SetCoordinates(i,nRadius,"Sin"));
                        CPen pen;
```

```
pen.CreatePen(PS_SOLID,3,GetSysColor(COLOR_3DHIGHLIGHT));
                if (i==0) dcMem->SelectObject(&pen);
                int Ss=dcMem->SetBkMode(TRANSPARENT);
                dcMem->Arc(x1,y1,x2,y2,x3,y3,x4,y4);
                dcMem->SetBkMode(Ss);
        }
}

// OnPaint
void CMeter::OnPaint()
{
        CPaintDC dc(this); // device context for painting
        CRect rcb ;
        GetClientRect(&rcb);
        CRect rc(rcb);//new rect
        rc=MyRect(rcb);//get myrect    - square
        int nPartRect=(int)rcb.bottom/5;

        CRect MyRect(rc); //deflate
        MyRect.left +=nPartRect;        MyRect.right -=nPartRect;
        MyRect.bottom -=nPartRect;    MyRect.top +=nPartRect;

        CDC dcMem;
        dcMem.CreateCompatibleDC(&dc);
        CBitmap bitmapMem;
        bitmapMem.CreateCompatibleBitmap(&dc, rc.Width(), rc.Height());
        CBitmap* pOldBit = dcMem.SelectObject(&bitmapMem);
        CBrush* pBr=&pBrush;
        dcMem.FillRect(rc,pBr); //square
        dcMem.SelectObject(&pPenFrame);
        dcMem.FillRect(MyRect,pBr);
        CRect rLI(DrawRect3D(&dcMem,rc));
        DeleteObject(pBr);
        CRect NewRect(MyRect);
        //section draw ellipse
        int nRadius=(int)(rc.right-rc.left)/2;
        int nRadiusY=(int)(rc.bottom-rc.top)/2;
        int nHight=(int)(NewRect.bottom-NewRect.top)/6;
        CRect r;
        r.left=nRadius-(int)nHight/2;    r.top=nRadiusY-(int)nHight/2;
        r.right=nRadius+(int)nHight/2;
r.bottom=nRadiusY+(int)nHight/2;
        CBrush* pOldBrush;
```

```
            pOldBrush=dcMem.SelectObject(&pBrushEllipse);
            dcMem.SelectObject(&pPenEllipse);
            dcMem.Ellipse(r);
            dcMem.SelectObject(pOldBrush);
              //end section draw ellipse
            //section draw scale
            CPen*pOldPen=dcMem.SelectObject(&pPenLow);
          nRadius=(int)(NewRect.right-NewRect.left)/2;
            int x1=NewRect.left;
            int y1=NewRect.top;
            int x2=NewRect.right;
            int y2=NewRect.bottom;
            CRect rect(rLI);
            for (int i=0;i<178;i++)
            {
                  int                                   x3=NewRect.left+(nRadius-
SetCoordinates(i+3,nRadius,"Cos"));
                  int                                   y3=NewRect.top+(nRadius-
SetCoordinates(i+3,nRadius,"Sin"));
                  int x4=NewRect.left+(nRadius-SetCoordinates(i,nRadius,"Cos"));
                  int y4=NewRect.top+(nRadius-SetCoordinates(i,nRadius,"Sin"));
                  if (i==0) dcMem.SelectObject(&pPenLow);
                  if (i>m_nGrad) dcMem.SelectObject(&pPenMiddle);
                  if (i>m_nGradMiddle+3)dcMem.SelectObject(&pPenHight);
                  DrawLineIndicator(&dcMem,rect,&pBrushLineIndicator,i);
                  dcMem.Arc(x1,y1,x2,y2,x3,y3,x4,y4);
            }

            //section draw scale
            dcMem.SelectObject(&pPenLineFont);
            dcMem.SelectObject(&pPenMiddle);
            DrawFillScale(&dcMem,NewRect,nRadius);
            DrawLineMeter(&dcMem,NewRect,nRadius);//it's Pie
            //section of Title text
            CFont*pOldFont;
            pOldFont=dcMem.SelectObject(SetMetTop.m_pFont);
            CPoint point;
            point.x=NewRect.left;
            //section toptitle
            point.y=NewRect.top-NewRect.Height()/4;
            m_colorText=SetMetTop.m_colorText;
            SetRectLR(&dcMem,NewRect,point,m_pStrTopTitle,SetMetTop.m_color
Bk);
            //section title
```

```
            dcMem.SelectObject(SetMetTitle.m_pFont);
            point.y=NewRect.top+NewRect.Height()/2+(int)(NewRect.Height()/4.5);
            m_colorText=SetMetTitle.m_colorText;
            SetRectLR(&dcMem,NewRect,point,m_pStrTitle,SetMetTitle.m_colorBk)
;
            //section of the Data
            dcMem.SelectObject(SetMetData.m_pFont);
            point.y +=(int)(NewRect.Height()/3.5);
            m_colorText=SetMetData.m_colorText;
            SetRectLR(&dcMem,NewRect,point,m_pStrData+"
",SetMetData.m_colorBk);
            //section of the begin scale
            dcMem.SelectObject(SetMetScale.m_pFont);
            point.x=NewRect.left-nPartRect-nPartRect/4;
            point.y=NewRect.top+NewRect.Height()/2+NewRect.Height()/20;
            CRect rcn(NewRect);
            rcn.right=NewRect.left+nPartRect;
            m_colorText=SetMetScale.m_colorText;
            SetRectLR(&dcMem,rcn,point," 0 ",SetMetScale.m_colorBk); //set "0" of
the begin
            //section of the End scale
            point.x=NewRect.right-nPartRect/3-nPartRect/2; //coordinates rect of a
text
            rcn.right=NewRect.right+nPartRect;
            CString pStr,pStr1;      pStr +=" ";
            pStr1 +=m_pStrMax+" ";
            pStr +=pStr1;
            SetRectLR(&dcMem,rcn,point,pStr,SetMetScale.m_colorBk);
            //end  sections
            dc.BitBlt(rc.left,rc.top,rc.Width(),rc.Height(),&dcMem,0,0,SRCCOPY);
            // Done with off-screen bitmap, DC and font     //
            dcMem.SelectObject(pOldFont);
            dcMem.SelectObject(pOldBit);
            dcMem.SelectObject(pOldPen);
            DeleteObject(bitmapMem.m_hObject);
            dcMem.DeleteDC();
        }


        // DMeterDlg.h : header file
        //
```

```
#if
!defined(AFX_DMETERDLG_H__CBE0666A_CF7C_40B7_9E76_206629773161__
INCLUDED_)
#define
AFX_DMETERDLG_H__CBE0666A_CF7C_40B7_9E76_206629773161__INCLUD
ED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "CMeter.h"
/////////////////////////////////////////////////////////////////////////////
// CDMeterDlg dialog


class CDMeterDlg : public CDialog
{
// Construction
public:
        CDMeterDlg(CWnd* pParent = NULL);     // standard constructor
    CMeter m_Meter,
                m_Meter2,
                m_Meter3;

// Dialog Data
        //{{AFX_DATA(CDMeterDlg)
        enum { IDD = IDD_DMETER_DIALOG };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDMeterDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);  //      DDX/DDV
support
        //}}AFX_VIRTUAL

// Implementation
protected:
        HICON m_hIcon;
        CFont m_pFontTitle,
                m_pFontTop,
                m_pFontScale,
```

```
            m_pFontData;

        // Generated message map functions
        //{{AFX_MSG(CDMeterDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        afx_msg void OnPaint();
        afx_msg HCURSOR OnQueryDragIcon();
        virtual void OnOK();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_DMETERDLG_H__CBE0666A_CF7C_40B7_9E76_206629773161__INCLUDED_)
// DMeterDlg.cpp : implementation file
//

#include "stdafx.h"
#include "DMeter.h"
#include "DMeterDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
```

```
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
        //}}AFX_VIRTUAL

    // Implementation
    protected:
        //{{AFX_MSG(CAboutDlg)
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
    };

    CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
    {
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
    }

    void CAboutDlg::DoDataExchange(CDataExchange* pDX)
    {
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
    }

    BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
            // No message handlers
        //}}AFX_MSG_MAP
    END_MESSAGE_MAP()

    /////////////////////////////////////////////////////////////////////////////
    // CDMeterDlg dialog

    CDMeterDlg::CDMeterDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CDMeterDlg::IDD, pParent)
    {
        //{{AFX_DATA_INIT(CDMeterDlg)
            // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
```

```
        // Note that LoadIcon does not require a subsequent DestroyIcon in
Win32
        m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
        m_pFontTop.CreatePointFont(160,"Arial");
        m_pFontTitle.CreatePointFont(135,"Arial");
        m_pFontScale.CreatePointFont(136,"Arial");
        m_pFontData.CreatePointFont(140,"Arial");
}

void CDMeterDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDMeterDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDMeterDlg, CDialog)
        //{{AFX_MSG_MAP(CDMeterDlg)
        ON_WM_SYSCOMMAND()
        ON_WM_PAINT()
        ON_WM_QUERYDRAGICON()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDMeterDlg message handlers

BOOL CDMeterDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        // Add "About..." menu item to system menu.

        // IDM_ABOUTBOX must be in the system command range.
        ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
        ASSERT(IDM_ABOUTBOX < 0xF000);

        CMenu* pSysMenu = GetSystemMenu(FALSE);
        if (pSysMenu != NULL)
        {
                CString strAboutMenu;
                strAboutMenu.LoadString(IDS_ABOUTBOX);
                if (!strAboutMenu.IsEmpty())
```

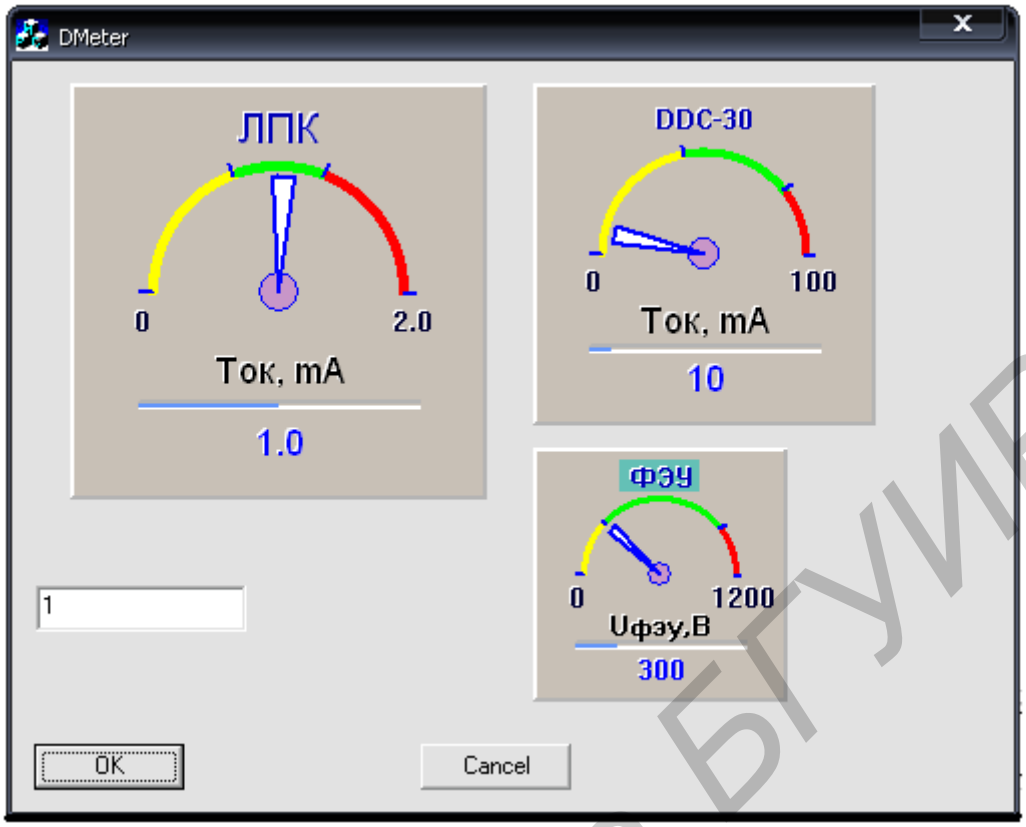
```
            {
                pSysMenu->AppendMenu(MF_SEPARATOR);
                pSysMenu->AppendMenu(MF_STRING,
IDM_ABOUTBOX, strAboutMenu);
            }
        }

        // Set the icon for this dialog.  The framework does this automatically
        //  when the application's main window is not a dialog
        SetIcon(m_hIcon, TRUE);            // Set big icon
        SetIcon(m_hIcon, FALSE);        // Set small icon

        if (m_Meter.Connect(IDC_METER,this)==FALSE) return FALSE;
        if (m_Meter2.Connect(IDC_METER2,this)==FALSE) return FALSE;
        if (m_Meter3.Connect(IDC_METER3,this)==FALSE) return FALSE;

        m_Meter.SetColorBk(RGB(200,192,182)); //frame
         //top
      m_Meter.SetMetTop.SetFont(&m_pFontTop);
        m_Meter.SetMetTop.SetColorText(RGB(0,0,182));
        m_Meter.SetMetTop.SetColorBkText(RGB(200,192,182));
        //title
        m_Meter.SetMetTitle.SetFont(&m_pFontTitle);
        m_Meter.SetMetTitle.SetColorText(RGB(0,0,0));
        m_Meter.SetMetTitle.SetColorBkText(RGB(200,192,182));
        //data
        m_Meter.SetMetData.SetFont(&m_pFontData);
        m_Meter.SetMetData.SetColorText(RGB(0,0,255));
        m_Meter.SetMetData.SetColorBkText(RGB(200,192,182));
        //scale
    //    m_Meter.SetMetScale.SetFont(&m_pFontScale);
        m_Meter.SetMetScale.SetColorText(RGB(0,0,50));
        m_Meter.SetMetScale.SetColorBkText(RGB(200,192,182));

        m_Meter.SetGrades(70,110);
        m_Meter.SetColorLineIndicator(RGB(100,150,250));
        m_Meter.SetTextTitle(" Ток, mA ");
        m_Meter.SetColorFontLine(RGB(0,0,255));
        m_Meter.GetMaxData(2,TRUE);
        m_Meter.SetTextTopTitle(" ЛПК ");
        m_Meter.GetDataMeter(0.5,TRUE);

        m_Meter2.SetColorBk(RGB(50,50,182)); //frame
         //top
```

```
//    m_Meter2.SetMetTop.SetFont(&m_pFontTop);
      m_Meter2.SetMetTop.SetColorText(RGB(0,0,182));
      m_Meter2.SetMetTop.SetColorBkText(RGB(200,192,182));
      //title
      m_Meter2.SetMetTitle.SetFont(&m_pFontTitle);
      m_Meter2.SetMetTitle.SetColorText(RGB(0,0,0));
      m_Meter2.SetMetTitle.SetColorBkText(RGB(200,192,182));
      //data
      m_Meter2.SetMetData.SetFont(&m_pFontData);
      m_Meter2.SetMetData.SetColorText(RGB(0,0,255));
      m_Meter2.SetMetData.SetColorBkText(RGB(200,192,182));
      //scale
//    m_Meter2.SetMetScale.SetFont(&m_pFontScale);
      m_Meter2.SetMetScale.SetColorText(RGB(0,0,50));
      m_Meter2.SetMetScale.SetColorBkText(RGB(200,192,182));
      //common
      m_Meter2.SetGrades(80,140);
      m_Meter2.SetColorLineIndicator(RGB(100,150,250));
      m_Meter2.SetTextTitle(" Ток, mA ");
      m_Meter2.SetColorFontLine(RGB(255,255,100));
      m_Meter2.GetMaxData(100,FALSE);
      m_Meter2.SetTextTopTitle(" DDC-30 ");
      m_Meter2.GetDataMeter(10,FALSE);

      m_Meter3.SetColorBk(RGB(200,192,182)); //frame
       //top
  //  m_Meter3.SetMetTop.SetFont(&m_pFontTop);
      m_Meter3.SetMetTop.SetColorText(RGB(0,0,182));
      m_Meter3.SetMetTop.SetColorBkText(RGB(100,192,182));
      //title
//    m_Meter3.SetMetTitle.SetFont(&m_pFontTitle);
      m_Meter3.SetMetTitle.SetColorText(RGB(0,0,0));
      m_Meter3.SetMetTitle.SetColorBkText(RGB(200,192,182));
      //data
//    m_Meter3.SetMetData.SetFont(&m_pFontData);
      m_Meter3.SetMetData.SetColorText(RGB(0,0,255));
      m_Meter3.SetMetData.SetColorBkText(RGB(200,192,182));
      //scale
//    m_Meter3.SetMetScale.SetFont(&m_pFontScale);
      m_Meter3.SetMetScale.SetColorText(RGB(0,0,50));
      m_Meter3.SetMetScale.SetColorBkText(RGB(200,192,182));

      m_Meter3.SetGrades(45,140);
      m_Meter3.SetColorLineIndicator(RGB(100,150,250));
```

```
        m_Meter3.SetTextTitle(" Uфэу,B ");
        m_Meter3.SetColorFontLine(RGB(0,0,255));
        m_Meter3.GetMaxData(1200,FALSE);
        m_Meter3.SetTextTopTitle(" ФЭУ ");
        m_Meter3.GetDataMeter(300,FALSE);



        // TODO: Add extra initialization here

        return TRUE;  // return TRUE  unless you set the focus to a control
}

void CDMeterDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
                CAboutDlg dlgAbout;
                dlgAbout.DoModal();
        }
        else
        {
                CDialog::OnSysCommand(nID, lParam);
        }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CDMeterDlg::OnPaint()
{
        if (IsIconic())
        {
                CPaintDC dc(this); // device context for painting

                SendMessage(WM_ICONERASEBKGND,          (WPARAM)
dc.GetSafeHdc(), 0);

                // Center icon in client rectangle
                int cxIcon = GetSystemMetrics(SM_CXICON);
                int cyIcon = GetSystemMetrics(SM_CYICON);
                CRect rect;
                GetClientRect(&rect);
```

```
            int x = (rect.Width() - cxIcon + 1) / 2;
            int y = (rect.Height() - cyIcon + 1) / 2;

            // Draw the icon
            dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
            CDialog::OnPaint();
        }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CDMeterDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}

void CDMeterDlg::OnOK()
{       CString pStr;
        CEdit* pEdit=(CEdit*)GetDlgItem(IDC_EDIT1);
        pEdit->GetWindowText(pStr);
   double res=atof(pStr);
        m_Meter.GetDataMeter(res,TRUE);
//      m_Meter2.GetDataMeter(res);
//      m_Meter.SetTextData(pStr);
//      m_Meter.SetColorText(RGB(200,0,0));

//      CDialog::OnOK();
}
```

Внешний вид вольтметра приведен на рисунке. При его создании использовались три объекта класса *CMeter*.



Внешний вид объекта

ЛИТЕРАТУРА

1. Шилд, Г. Программирование на Borland C++ для профессионалов / Г. Шилд. – Минск : ООО Попурри, 1998.

2. Пол, И. Объектно-ориентированное программирование с использованием C++ / И. Пол. – Киев : НПИФ ДиаСофт, 1995.

3. Программирование на Visual C++ 6.0 для профессионалов / Д. Круглински [и др.]. – СПб.: Питер, 2004.

4. Смолякова, О. Лабораторный практикум по объектно-ориентированному программированию / О. Смолякова. – Минск : БГУИР, 2005.

СОДЕРЖАНИЕ

Св. план 2011, резерв

Учебное издание

КЛАССЫ И ОБЪЕКТЫ В ЯЗЫКЕ С++

Методические указания
для практических занятий по курсу
«Объектно-ориентированное программирование»
для студентов специальностей 1-45 01 03 «Сети телекоммуникаций»,
1-45 01 05 «Системы распределения мультимедийной информации»,
1-98 01 02 «Защита информации в телекоммуникациях»
дневной и заочной форм обучения

Составители:
Курейчик Константин Петрович
Мельников Владимир Александрович

Редактор Н. В. Гриневич
Корректор Е. Н. Батурчик

Подписано в печать 25.04.2012.	Формат 60x84 1/16.	Бумага офсетная.
Гарнитура «Таймс».	Отпечатано на ризографе.	Усл. печ. л. 2,67.
Уч.-изд. л. 1,8.	Тираж 50 экз.	Заказ 695.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009.
220013, Минск, П. Бровки, 6