В.Л. Тарасов

Лекции по программированию на C++

Лекция 11

Классы

С помощью структур, рассмотренных выше, можно собрать в одно целое несколько данных, описывающих какой-либо предмет или явление. Функции, включаемые в состав структуры, упрощают доступ к данным, содержащимся в структуре. У структур есть недостаток - данные, хранящиеся в структуре, доступны не только функциям-членам структуры, но и независимым функциям, то есть структуры делают более удобной работу с данными, но не защищают их от изменения сторонними функциями. Это затрудняет поиск ошибок при неправильной обработке данных, хранящихся в структуре.

Для большей защиты данных используют классы, которые подобны структурам, но предоставляют возможность *скрывать* данные от всех функций, кроме функций-членов класса.

Переменные, имеющие тип структуры или класса часто называют *объектами*, а программирование с использованием классов называется *объектно-ориентированным*.

11.1. Классы. Скрытие данных

Объявление классов похоже на объявление структур, но производится с использованием ключевого слова `class`. По умолчанию все члены класса, в отличие от членов структуры, являются *закрытыми*, то есть недоступными для сторонних функций, не являющихся членами данного класса.

Перепишем программу 9.4, применив для моделирования времени суток не структуру, а класс.

Программа 11.1. Время суток как класс

Определение класса для работы со временем суток поместим в файл `TimeDayClass.h`. Далее приведено содержимое этого файла.

```
// Файл TimeDayClass.h
#ifndef TimeDayClassH
```

```
#define TimeDayClassH

#include <iostream>
#include <iomanip>       // Для доступа к функциям-манипуляторам
#include <cstdlib>
using namespace std;

class TimeDay{           // Класс для моделирования времени суток
    int hour;            // Часы суток
    int min;             // Минуты
public:                  // Раздел открытых членов класса
    void Set(int hh, int mm)         // Установка времени
    {hour = hh; min = mm;}
    void AddHour(int nh);            // Добавить nh часов
    void AddMin(int nm);             // Добавить nm минут
    void Print();                    // Вывод времени
};
#endif
```

Есть два отличия в определении класса `TimeDay` от одноименной структуры из файла `TimeDayFunc.h` программы 9.4: слово `struct` заменено на слово `class`, и в классе имеется метка `public`. По умолчанию все члены класса *закрыты*, открытыми являются члены класса, объявленные после метки `public`. К закрытым членам класса могут обращаться только функции-члены класса. Внешние по отношению к классу функции, например, `main()` могут обращаться только к открытым членам класса.

Определения функций-членов класса поместим в файле `TimeDayClass.cpp`. Его содержимое полностью совпадает с содержимым файла `TimeDayFunc.cpp`.

```
// Файл TimeDayClass.cpp

#include "TimeDayClass.h"

void TimeDay::AddHour(int nh)         // Добавить nh часов
{
    hour = (hour + nh) % 24;
}

void TimeDay::AddMin(int nm)          // Добавить nm минут
{
    hour = (hour + (min + nm) / 60) % 24;
    min = (min + nm) % 60;
}

void TimeDay::Print()                 // Вывод времени
{
    cout << setw(2) << setfill('0') << hour << ':'
         << setw(2) << setfill('0') << min << ' ';
}
```

Таким образом, функции-члены класса определяются так же как функции-члены структуры.

Для испытания класса TimeDay используем главную функцию, которая почти совпадает с аналогичной функцией для тестирования структуры TimeDay (см. файл UseTimeDay.cpp в программе 9.4).

```
// Файл UseTimeDayClass.cpp

#include "TimeDayClass.h"

int main()
{
    setlocale(LC_ALL, "Russian");
    int hpair        = 1;               // Длительность пары: часы
    int mpair        = 30;              //                    минуты
    int interval = 10;                  // Длительность перерыва в минутах
    int n = 6;                          // Количество пар
    TimeDay tstart;                     // Время начала занятий
    tstart.Set(8, 00);                 // Установка времени начала занятий
    TimeDay pair = tstart;             // Установка времени в структуре Pair
    cout << "Начало Конец" << endl;     // Заголовок таблицы
    for(int i = 0; i < n; i++){
        pair.Print();                   // Печать времени начала пары
        pair.AddHour(hpair);            // Расчет времени
        pair.AddMin(mpair);             // конца пары
        pair.Print(); cout << endl;     // Вывод времени конца пары
        pair.AddMin(interval);          // Расчет начала следующей пары
    }
    TimeDay morning;                    // Время подъема
    //morning.hour = 6;
    //morning.min = 30;
    morning.Set(6, 30);                 // Установка значения для morning
    cout << "Подъем утром в ";
    morning.Print();                    // Вывод значения morning
    cout << endl;
    system("pause");
    return 0;
}
```

Вывод программы такой же, как программы 6.4:

```
Начало Конец
08:00 09:30
09:40 11:10
11:20 12:50
13:00 14:30
14:40 16:10
16:20 17:50
Подъем утром в 06:30
```

Обратим внимание на две закомментированные строки в файле UseTimeDayClass.cpp:

```
//morning.hour = 6;
//morning.min = 30;
```

Если убрать комментарии, возникнет ошибка из-за попытки обратиться к *закрытым* членам класса `TimeDay::hour` и `TimeDay::min`:

```
Ошибка  1   error C2248: TimeDay::hour: невозможно обратиться к private
член, объявленному в классе "TimeDay"
Ошибка  2   error C2248: TimeDay::min: невозможно обратиться к private член,
объявленному в классе "TimeDay"
```

В описании ошибки использовано слово `private`, которое является ключевым словом языка, применяемым для обозначения закрытых членов класса. Хотя в классе все члены являются по умолчанию закрытыми, такие члены можно задавать явно с помощью ключевого слова `private`. Например,

```
class TimeDay{
private:
    int hour;
    int min;
public:
    void Set(int hh, int mm);
    void AddHour(int nh);
    void AddMin(int nm);
    void Print();
};
```

Здесь явно указано, что `hour` и `min` – закрыты.

Разделов `public` и `private` в объявлении класса может быть несколько, и они могут идти в произвольном порядке:

```
class TimeDay{
private:
    int hour;
public:
    void Set(int hh, int mm);
    void AddHour(int nh);
private:
    int min;
public:
    void AddMin(int nm);
    void Print();
};
```

В языке C++ структуры отличаются от классов только тем, что все члены структуры открыты по умолчанию. Часть членов структуры можно сделать закрытыми с помощью ключевого слова `private`. Таким образом, структуру `TimeDay` из программы 9.4 можно сделать тождественной классу `TimeDay` из программы 11.1 объявив ее в виде:

```
struct TimeDay{          // Структура для моделирования времени суток
```

```
private:
    int hour;
    int min;
public:
    void Set(int hh, int mm)
    {hour = hh; min = mm;}
    void AddHour(int nh);
    void AddMin(int nm);
    void Print();
};
```

Теперь члены `hour` и `min` стали закрытыми, и при компиляции `main()` из программы 9.4 будет обнаружена ошибка:

```
    morning.hour = 6;    // Ошибка, попытка обратиться к закрытому члену
    morning.min = 30;    // Ошибка, попытка обратиться к закрытому члену
```

11.2. О системном времени

Для получения текущей даты и текущего времени в программах на C++ можно воспользоваться библиотекой с заголовочным файлом `ctime`. Библиотечные функции отсчитывают время в секундах от начала суток 1 января 1970 года. Для секунд используется тип `time_t`, определенный инструкцией:

```
typedef long time_t;     // Вводится наименование time_t для типа long
```

Библиотечная функция

```
time_t time(time_t* pseconds);
```

возвращает количество секунд, прошедших с полуночи (00:00:00) 1 января 1970 всемирного скоординированного времени (Coordinated Universal Time, UTC), по системным часам[1]. Возвращаемое значение сохраняется также по адресу, указанному параметром `pseconds`. Если этот параметр равен `NULL`, возвращаемое значение не сохраняется.

Период времени, начавшийся с полуночи 01.01.1970, называется эпохой Юникс. В стандартной библиотеке для хранения секунд используется переменная типа `long`, которая имеет максимальное значение $2^{31} - 1 = 2\,147\,483\,647$. Счетчик секунд достигнет этого предельного значения 19 января 2038 года в 03:14:07 по UTC. После добавления еще одной секунды значение счетчика станет

[1] Время UTC введено в 1964 г. для уточнения среднего временем по Гринвичу (Greenwich Mean Time, GMT). Время в других часовых поясах отсчитывается от UTC (практически – от гринвичского). Московское время опережает время UTC на 3 часа.

отрицательным и равным -2 147 483 648, а это значение соответствует 13 декабря 1901, 8:45:52. Таким образом, программы, использующие 4 байта для хранения времени, после 2038 года могут работать неверно. Учитывая это, в Visual C++ функция time сделана «оберткой» для 64 разрядной функции _time64, а тип time_t по умолчанию эквивалентен 64 разрядному типу __time64_t. Если нужно заставить компилятор интерпретировать time_t как старое 32-битное time_t, можно определить макрос _USE_32BIT_TIME_T, однако это не рекомендуется, так как приложение может неправильно работать после 18 января 2038 года.

В заголовке ctime объявлена структура tm, используемая при работе с датами и временем:

```
struct tm{
   int tm_sec;          /* Секунды, 0-59 */
   int tm_min;          /* Минуты, 0-59 */
   int tm_hour;         /* Часы (0 - 23) */
   int tm_mday;         /* День месяца (1 - 31) */
   int tm_mon;          /* Месяц (0 - 11) */
   int tm_year;         /* Год (календарный год минус 1900) */
   int tm_wday;         /* День недели, начиная с воскресенья (0 - 6) */
   int tm_yday;         /* День года (0 - 365) */
   int tm_isdst;        /* Летнее время: > 0 - действует,
                        0 – не действует, < 0 – не определено */
};
```

Функция

```
char* asctime(const struct tm* ptm);
```

извлекает компоненты даты и времени из структуры *ptm, размещает их в статически выделенной строке в форме:

```
день месяц число часы:минуты:секунды год\n\0
```

и возвращает указатель на эту строку.

Функция

```
char* ctime(const time_t* pseconds);
```

принимает указатель pseconds на переменную, содержащую время в секундах, и возвращает указатель на строку вида:

```
день месяц число часы:минуты:секунды год\n\0
```

Функция

```
struct tm* gmtime(const time_t* pseconds);
```

возвращает указатель на структуру tm, содержащую компоненты даты и времени в виде всемирного скоординированного времени. Если система не поддерживает всемирное время, возвращается значение NULL.

Структура, используемая функцией gmtime() для хранения времени, выделяется статически и перезаписывается при каждом вызове этой функции.

Функция

```
struct tm* localtime(const time_t* pseconds);
```

возвращает указатель на структуру tm, содержащую компоненты даты и локального времени. Аргумент pseconds указывает на переменную, содержащую секунды, прошедшие от установленной точки отсчета.

Функция

```
time_t maketime(struct tm * ptm);
```

возвращает время в виде секунд по структуре, содержащей компоненты даты и времени, на которую указывает ptm. Члены структуры tm_wday; tm_yday задавать не нужно, они устанавливаются самой функцией. Если календарное время не может быть представлено, возвращает -1.

С помощью функции

```
clock_t clock(void);
```

можно определить время работы программы. Здесь использовано обозначение:

```
typedef long clock_t;
```

Для преобразования возвращаемого функцией значения в секунды, его надо поделить на константу CLOCKS_PER_SEC, которая определена в заголовке ctime и равна количеству «тиков» процессора в одну секунду. Если запрашиваемое время недоступно, clock(void) возвращает -1.

Программа 11.2. Работа с системным временем

В программе демонстрируется использование библиотеки с заголовочным файлом ctime для работы со временем.

```
#include <ctime>
#include <iostream>
#include <string>
using namespace std;

int main()
{
   clock_t start, end;
   start = clock();      // Фиксируем время начала работы программы

   string Months[] = {" января", " февраля", " марта", " апреля",
                      " мая", " июня", " июля", " августа",
                      " сентября", " октября", " ноября",
                      " декабря"};
```

```
   string DayWeek[] = {"воскресенье ", "понедельник ", "вторник ",
                       "среда ", "четверг ", "пятница ", "суббота "};
   setlocale(LC_ALL, "Russian");

   tm* ptm_time;              // Указатель на структуру для времени
   time_t seconds;            // Переменная для секунд
   time(&seconds);            // Получение текущего времени в секундах

   cout << "Время UTC в секундах, начиная с 01.01.1970:\t"
        << seconds << endl;
   cout << "Местные дата и время:\t\t\t" << ctime(&seconds);

   ptm_time = localtime(&seconds);    // Преобразование секунд в структуру
   cout << "Сегодня " << DayWeek[ptm_time->tm_wday]    // День недели
        << ptm_time->tm_mday                           // Число (день месяца)
        << Months[ptm_time->tm_mon] << " "                      // Месяц
        << (ptm_time->tm_year + 1900) << " года. Сейчас  "     // Год
        << ptm_time->tm_hour << " часов "
        << ptm_time->tm_min  << " минут "
        << ptm_time->tm_sec  << " секунд\n";

   seconds = 2147483647;                               // Конец эпохи Юникс
   cout << "КОНЕЦ эпохи Юникс (местное время): \t" << ctime(&seconds);
   cout << "КОНЕЦ эпохи Юникс (время UTC):\t\t"
        << asctime(gmtime(&seconds));

// Попытка поработать с датой до эпохи Юникс
   tm  Gagarin = { 0, 7, 9, 12, 3, 61};   // 9 часов 7 минут 12.04.1961
   if( mktime(&Gagarin) != time_t(-1))
      cout << "Полет Гагарина: \t\t" << asctime(&Gagarin);
   else
      cout << "Не удалось сформировать время до начала эпохи Юникс\n";

// Последняя секунда эпохи Юникс по московскому времени
   tm  LastSecondEpochUnix = { 7, 14, 6, 19, 0, 2038 - 1900};
   if( mktime(&LastSecondEpochUnix) != time_t(-1))
      cout << "Последняя секунда эпохи Юникс: \t\t"
           << asctime(&LastSecondEpochUnix);
   else
      cout << "Не удалось создать время последней секунды эпохи Юникс\n";

// Первая секунда после эпохи Юникс по московскому времени
   struct tm  FirstSecondAfterEpochUnix = { 8, 14, 6, 19, 0, 2038 - 1900};
   if( mktime(&FirstSecondAfterEpochUnix) != time_t(-1))
      cout << "Время после эпохи Юникс: \t\t"
           << asctime(&FirstSecondAfterEpochUnix);
   else
      cout << "Не удалось сформировать время после эпохи Юникс\n";
   end = clock();        // Фиксируем время конца работы программы

   cout << "Программа работала " << (end - start) << " тиков или "
        << (end - start) / CLOCKS_PER_SEC << " секунд\n";
   cin.get();
   return 0;
}
```

Программа вывела:

```
Время UTC в секундах, начиная с 01.01.1970:     1449681758
Местные дата и время:                  Wed Dec 09 20:22:38 2015
Сегодня среда 9 декабря 2015 года. Сейчас  20 часов 22 минут 38 секунд
КОНЕЦ эпохи Юникс (местное время):     Tue Jan 19 06:14:07 2038
КОНЕЦ эпохи Юникс (время UTC):         Tue Jan 19 03:14:07 2038
Не удалось сформировать время до начала эпохи Юникс
Последняя секунда эпохи Юникс:         Tue Jan 19 06:14:07 2038
Время после эпохи Юникс:               Tue Jan 19 06:14:08 2038
Программа работала 31 тиков или 0 секунд
```

11.3. Конструкторы

Для инициализации объектов класса создаются специальные функции-члены – *конструкторы*. Имя конструктора совпадает с именем класса. Конструктор не возвращает никакого значения, даже void. Если класс имеет конструктор, то конструктор вызывается *автоматически* при создании объекта класса, поэтому все объекты класса будут проинициализированы. Конструкторов может быть несколько, они различаются по правилам перегруженных функций, то есть по числу и типам аргументов.

Дополним класс TimeDay из программы 11.1 конструкторами.

Программа 11.3. Конструктор в классе время дня

```
// Файл TimeDayConstr.h

#ifndef TimeDayConstrH
#define TimeDayConstrH

#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;

class TimeDay{                // Класс для времени суток
    int hour;                 // Часы суток
    int min;                  // Минуты часа
public:                       // Раздел открытых членов класса
    TimeDay(int hh, int mm)   // Конструктор
    { hour = hh; min = mm; } // Установка времени
    TimeDay()                 // Конструктор без аргументов (по умолчанию)
    { hour = 0; min = 0; }    // Начало суток
    void Print();             // Вывод времени
};

#endif
```

Теперь при создании переменных типа TymeDay будет автоматически вызываться конструктор, например:

```
TimeDay StartWork(8, 0);
```

Аргументы конструктора указываются в скобках после имени создаваемой переменной. Конструктор присвоит членам StartWork значения: StartWork.hour = 8, StartWork.min = 0.

Если при создании объекта класса не указаны аргументы конструктора, будет вызван конструктор без аргументов (конструктор по умолчанию), если он есть в классе. Например,

```
TimeDay StartDay;
```

Компоненты StartDay будут: StartDay.hour = 0, StartDay.min = 0 - это время начала суток.

В состав класса входит функция для вывода даты. Ее определение находится в файле:

```
// Файл TimeDayConstr.cpp

#include "TimeDayConstr.h"
void TimeDay::Print()                // Вывод времени
{
    cout << setw(2) << setfill('0') << hour << ':'
         << setw(2) << setfill('0') << min << ' ';
}
```

Для тестирования класса TimeDay, имеющего теперь конструктор, используем программу:

```
// Файл UseTimeDayConstr.cpp

#include "TimeDayConstr.h"

int main()
{
    setlocale(LC_ALL, "Russian");
    TimeDay StartDay;            // Начало суток.
                                 // Использование конструктора по умолчанию
    cout << "Начало дня:\t";     StartDay.Print();
    TimeDay StartWork(8, 0);         // Использование
    TimeDay EndWork(17, 30);         // конструктора с параметрами
    cout << "\nНачало работы:\t"; StartWork.Print();
    cout << "\nКонец работы:\t";  EndWork.Print();
    cout << endl;
    system("pause");
    return 0;
}
```

Программа выводит:

```
Начало дня:     00:00
Начало работы:  08:00
Конец работы:   17:30
```

Можно было бы назначить другие значения по умолчанию для аргументов конструктора, например:

```
TimeDay(int hh = 17, int mm = 30) // Конструктор
{ hour = hh; min = mm; }          // Установка времени
```

Тогда при создании переменной в виде:

```
TimeDay EndWork;
```

ее значениями будут: `EndWork.hour = 17`, `EndWork.min = 30`.

11.4. Статические члены и размер класса

Статические члены класса

При создании объектов класса каждый объект получает свой полный набор данных-членов класса. Но бывают ситуации, когда надо иметь некоторые данные в одном экземпляре на целый класс, независимо от того, сколько объектов класса создается. Такие, уникальные на весь класс данные объявляются в классе как статические члены с ключевым словом `static`.

Функции, предназначенные для работы со статическими членами класса, объявляются также с ключевым словом `static`.

Пример создания и использования статических членов класса приводится в следующей программе.

Программа 11.4. Статические члены класса

Модифицируем класс для времени суток `TimeDay`, добавив в него статические члены для времени обеденного перерыва и статические функции для работы с этими статическими членами.

```
// Файл TimeDayStatic.h

#ifndef TimeDayStaticH
#define TimeDayStaticH

#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;

class TimeDay{                          // Время суток
    int hour;                           // Часы
    int min;                            // Минуты часа
    static int hdin, mdin;              // Статические члены: время обеда
public:
    TimeDay(int hh = 0, int mm = 0)     // Конструктор
    {hour = hh; min = mm;}              // Установка времени
```

```
    void Print();                    // Вывод времени
    static void SetDinner(int hd, int md)    // Статическая функция:
    { hdin = hd; mdin = md; }                // Установка времени обеда
    static void PrnDinner()                  // Вывод времени обеда
    { TimeDay din(hdin, mdin); din.Print(); }
};
#endif
```

Здесь у конструктора указаны значения аргументов по умолчанию. Это позволило заменить два конструктора предыдущей программы одним.

Статические переменные-члены класса в классе только *объявляются*, то есть под них не выделяется память. Поэтому их надо где-либо в программе определить. Это сделано в следующем файле.

```
// Файл TimeDayStatic.cpp

#include "TimeDayStatic.h"

void TimeDay::Print()                // Вывод времени
{
    cout << setw(2) << setfill('0') << hour << ':'
         << setw(2) << setfill('0') << min << ' ';
}

// Определение статических членов класса
// Здесь под статические члены выделяется память

int TimeDay::hdin, TimeDay::mdin;
```

Обычные функции-члены класса вызываются для объектов класса. Статические функции-члены работают на класс в целом, поэтому при их вызове можно указывать класс. Это демонстрируется в следующей программе.

```
// Файл UseTimeDayStatic.cpp

#include "TimeDayStatic.h"
int main()
{
    setlocale(LC_ALL, "Russian");

    TimeDay StartWork(8, 0);
    cout << "Начало работы:\t";
    StartWork.Print();

    // Вызов статической функции-члена класса
    TimeDay::SetDinner(12, 30);      // Установка времени обеда

    cout << "\nОбед:\t\t";
    TimeDay::PrnDinner();            // Вывод времени обеда

    cout << endl;
    system("pause");
    return 0;
```

```
}
```

Программа выводит:

```
Начало работы:  08:00
Обед:           12:30
```

Размер класса и объектов класса

Функции-члены класса существуют в одном экземпляре на весь класс и не влияют на размер объектов класса. Убедиться в этом можно с помощью следующей программы.

Программа 11.5. Размер класса и объектов класса

Создадим проект, в который включим заголовочный файл `SizeTimeDay.h`, содержимое которого совпадает с содержимым файла `TimeDayStatic.h` из предыдущей программы 11.4.

Создадим в проекте файл `SizeTimeDay.cpp` совпадающий с файлом `TimeDayStatic.cpp`.

В функции `main()` выведем размер класса `TimeDay` и размер объектов этого класса.

```
// Файл ShowSizeTimeDay.cpp

#include "SizeTimeDay.h"
int main()
{
    setlocale(LC_ALL, "Russian");
    cout << "Размер класса TimeDay = " << sizeof(TimeDay) << endl;

    TimeDay StartWork(8, 0);                    // Объект класса
    cout << "Размер объекта класса TimeDay = " << sizeof(StartWork) << endl;
    system("pause");
    return 0;
}
```

Программа выводит:

```
Размер класса TimeDay = 8
Размер объекта класса TimeDay = 8
```

Видно, что размер класса и размер объектов класса одинаковы и равны 8 байт, так как класс состоит из двух переменных типа `int`, а в той системе, где выполнялась программа, размер `int` равен 4 байтам. Функции-члены класса не влияют на размер объектов класса. Они существуют в единственном экземпляре на весь класс. Принадлежность функции к классу означает лишь возможность ее доступа к членам класса без явного указания этих членов как аргументов функции.

Статические переменные статические функции существуют также в единственном экземпляре на весь класс и не влияют на размер объектов класса.

11.5. Друзья класса

Иногда требуется открыть доступ к закрытым членам класса для функций, которые не являются членами класса. Такие внешние функции надо объявить в классе с ключевым словом `friend` (друг). В следующей программе другом является функция:

```
friend int before(const TimeDay& t1, const TimeDay& t2);
```

которая обращается к закрытым членам класса и выясняет, какой момент времени `t1` или `t2` раньше.

В класс включены два конструктора: конструктор с параметрами устанавливает заданное время, конструктор по умолчанию устанавливает текущее время по системным часам.

Программа 11.6. Друг класса для сравнения времени

Объявим класс для времени в виде:

```
// Файл FriendTimeDay.h

#ifndef FriendTimeDayH
#define FriendTimeDayH

#include <iostream>
#include <iomanip>
#include <ctime>
#include <cstdlib>
using namespace std;

class TimeDay{                          // Время суток
    int hour;                           // Часы
    int min;                            // Минуты часа
    static int hdin, mdin;              // Статические члены: время обеда
public:
    TimeDay(int hh, int mm )      // Конструктор с аргументами
    { hour = hh; min = mm;}
    TimeDay();                    // Конструктор по умолчанию, без аргументов
    void Print();                       // Вывод времени

// Объявления функции-друга класса для сравнения двух моментов времени
    friend int before(const TimeDay& t1, const TimeDay& t2);
};
#endif
```

По сравнению с программой 11.4 в класс не включены статические члены, явно объявлены два конструктора – с параметрами и по умолчанию.

Функция-друг класса before() объявлена внутри класса с ключевым словом friend.

Реализация конструктора и дружественной функции находятся в файле:

```
// Файл FriendTimeDay.cpp

#include "FriendTimeDay.h"

void TimeDay::Print()                  // Вывод времени
{
    cout << setw(2) << setfill('0') << hour << ':'
             << setw(2) << setfill('0') << min << ' ';
}

// before: возвращае значение -1, если время t1 < t2,
// 0 при t1==t2 и 1 при t1 > t2
int before(const TimeDay& t1, const TimeDay& t2)
{
    if(t1.hour < t2.hour || (t1.hour == t2.hour && t1.min < t2.min))
        return -1;
    else if(t1.hour == t2.hour && t1.min == t2.min)
        return 0;
    else
        return 1;
}

// Конструктор по умолчанию устанавливает текущее время
TimeDay::TimeDay()
{
    tm* ptm_time;                      // Указатель на структуру tm
    time_t seconds = time(0);          // Число секунд от начала отсчета
    ptm_time = localtime(&seconds);    // Преобразование секунд в дату
    hour = ptm_time->tm_hour;          // Часы
    min = ptm_time->tm_min;            // Минуты
}
```

В главной функции тестируется созданный класс.

```
// Файл UseFriendTimeDay.cpp

#include "FriendTimeDay.h"
int main()
{
    setlocale(LC_ALL, "Russian");
    TimeDay EndWork(17, 30);
    cout << "Конец работы:  "; EndWork.Print();
    TimeDay Curr;                          // Текущее время устанавливается
                                           // конструктором по умолчанию
    cout << "\nСейчас: "; Curr.Print();
```

```
    if((int res = before(Curr, EndWork)) < 0)     // Вызов функции-друга
        cout << "Рабочее время не окончилось\n";  // before()
    else if(res == 0)
        cout << "Конец работы\n";
    else
        cout << "Нерабочее время\n";
    system("pause");
    return 0;
}
```

Программа выдает следующее:

```
Начало работы: 08:30
Конец работы:  17:30
Сейчас: 11:14 Нерабочее время
```

11.6. Копирование объектов класса

Допустимы *инициализация* объекта класса другим объектом того же класса и *присваивание* объектов. При выполнении инициализации и присваивания происходит *почленное* копирование. В следующей программе на примере класса `TimeDay` демонстрируются инициализация и копирование объектов.

Программа 11.7. Копирование объектов

Возьмем за основу предыдущую программу 11.6.

Создадим проект, в который включим заголовочный файл `FriendTimeDay.h` и файл исходного кода `FriendTimeDay.cpp` из предыдущей программы. В отдельном файле разместим функцию `main()`. Несмотря на то, что заголовочный `FriendTimeDay.h` вставлен в проект, при его включении в другие файлы нужно указывать его полное имя с указанием папки, чтобы препроцессор мог его найти не в текущей папке проекта.

```
// Файл UseFriendTimeDay.cpp

#include "..\\Progr_11_06_TimeDayFriend\\FriendTimeDay.h"

int main()
{
    setlocale(LC_ALL, "Russian");
    TimeDay EndWork(17, 30); // EndWork инициализирует конструктор

    cout << "Конец работы, план: "; EndWork.Print();

    TimeDay Now;             // Now инициализирует конструктор по умолчанию
    TimeDay Curr = Now;      // Curr инициализируется Now

    cout << "\nСейчас: "; Curr.Print();

    EndWork = Now;           // Присваивание объектов
```

```
    cout << "\nКонец работы, факт: "; EndWork.Print();
    cout << endl;
    system("pause");
    return 0;
}
```

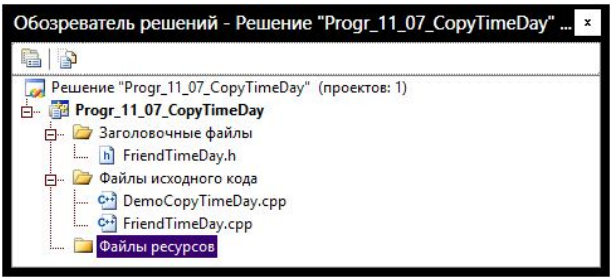

Рис. 11.1. Состав проекта

Программа выводит:

```
Конец работы, план: 17:30
Сейчас: 09:55
Конец работы, факт: 09:55
```

Если в классе есть конструктор, он выполняет инициализацию создаваемого объекта. В рассматриваемой программе конструктор с параметрами инициализирует `EndWork`, конструктор по умолчанию инициализирует `Now`.

Инструкция:

```
TimeDay Curr = Now;
```

создает новый объект `Curr` и делает его копией `Now`.

Присваивание вида:

```
EndWork = Now;
```

производится на *этапе выполнения* программы. При присваивании также производится почленное копирование, поэтому `EndWork` становится копией `Now`.

11.7. Ссылка на себя

Каждая нестатическая функция-член «знает», для какого объекта она вызвана и может явно на него ссылаться. Для того использую ключевое слово `this`, которое является указателем на объект, для которого вызвана функция.

Программа 11.8. Модификация времени

В данной программе в класс TimeDay включены функции-члены для модификации времени, которые возвращают ссылку на измененное время.

```
// Файл ModifTime.h

#ifndef ModifTimeH
#define ModifTimeH

#include <iostream>
#include <iomanip>
#include <ctime>
#include <cstdlib>
using namespace std;

class TimeDay{                          // Время суток
    int hour;                           // Часы
    int min;                            // Минуты часа
public:
    TimeDay(int hh, int mm )            // Конструктор
    { hour = hh; min = mm;}
    void Print();                       // Вывод времени

    TimeDay& AddHour(int h)             // Увеличить время на h часов
    {
        hour = (hour + h) % 24;         // Изменение часов
        return *this;                   // Возврат ссылка на измененное время
    }

    TimeDay& AddMin(int m)                      // Увеличить время на m минут
    {
        hour = (hour + (min + m) / 60) % 24;    // Изменение часов
        min = (min + m) % 60;                   // Изменение минут
        return *this;
    }

    TimeDay& AddTime(const TimeDay& td)         // Увеличить на время td
    {
        AddHour(td.hour);                       // Добавление часов
        AddMin(td.min);                         // Добавление минут
        return *this;
    }
};

#endif
```

Ключевое слово this внутри функции имеет значение адреса того объекта класса TimeDay, для которого функция вызвана. Например, далее в main() есть инструкция вызова функции AddHour() для объекта MyTime:

```
MyTime.AddHour(3);
```

Во время работы AddHour() указатель this имеет значение адреса переменной MyTime, а выражение *this есть сама переменная MyTime. Инструкция

```
return *this;
```

возвращает ссылку на переменную, для которой вызвана AddHour(), то есть ссылку на MyTime.

```
// Файл ModifTime.cpp

#include "ModifTime.h"
void TimeDay::Print()                  // Вывод времени
{
    cout << setw(2) << setfill('0') << hour << ':'
             << setw(2) << setfill('0') << min << ' ';
}

// Файл TestModifTime.cpp

#include "ModifTime.h"
int main()
{
    setlocale(LC_ALL, "Russian");
    TimeDay MyTime(7, 25);       //
    cout << "MyTime = "; MyTime.Print();

    MyTime.AddHour(3);
    MyTime.AddMin(50);
    cout << "\nЧерез 3 ч 50 мин MyTime = "; MyTime.Print();

    TimeDay AnyTime(2, 45);
    cout << "\nAnyTime = "; AnyTime.Print();

    MyTime.AddTime(AnyTime);
    cout << "\nЕще через 2 ч 45 мин MyTime = "; MyTime.Print();

    MyTime.AddMin(10).AddHour(1).AddTime(AnyTime);

    cout << "\nЕще через 10 мин 1 ч и 2 ч 45 мин MyTime = ";
    MyTime.Print();
    system("pause");
    return 0;
}
```

Программа выводит:

```
MyTime = 07:25
Через 3 ч 50 мин MyTime = 11:15
AnyTime = 02:45
Еще через 2 ч 45 мин MyTime = 14:00
Еще через 10 мин 1 ч и 2 ч 45 мин MyTime = 17:55
```

Когда в выражение входят несколько операторов «точка» (.), они выполняются слева направо, то есть инструкция

```
MyTime.AddMin(10).AddHour(1).AddTime(AnyTime);
```

эквивалентна следующей:

```
((MyTime.AddMin(10)).AddHour(1)).AddTime(AnyTime);
```

Выражение `MyTime.AddMin(10)` равно времени `MyTime`, увеличенному на 10 минут, так как `AddMin()` возвращает *ссылку* на объект, для которого вызывается. Далее для этого измененного `MyTime` вызывается функция `AddHour()` которая увеличивает время на 1 час. Затем для этого измененного времени вызывается `AddTime()` и увеличивает время на `AnyTime`. Возможность писать подобные цепочки вызовов функций обеспечивается тем, что функции возвращают *ссылки* на объекты.

11.8. Графическая библиотека OpenGL

Данный параграф не относится к теме «классы». Здесь дается краткое описание графической библиотеки OpenGL, с использованием которой ряд примеров программ можно сделать более наглядными.

OpenGL (Open Graphics Library — открытая графическая библиотека) включает более 300 функций для рисования сложных трёхмерных сцен из простых примитивов. Официальным описанием OpenGL является «красная книга», электронный вариант которой свободно доступен для скачивания и просмотра [1]. Имеется перевод «красной книги» на русский язык [2]. Файл динамически загружаемой библиотеки `opengl32.dll`, необходимой для работы приложений, использующих OpenGL, поставляются в составе Windows. Для 32-разрядной версии Windows он расположен в папке `Windows\System32`, для 64-разрядной - в папке `Windows\SysWOW64`.

Для облегчения использования OpenGL был разработан набор утилит `GLUT` (от `GL Utility Toolkit`) [3]. Библиотека `GLUT` не является проектом с открытым исходным кодом и давно не обновлялась. Есть ее полностью свободный вариант - `freeGLUT` [4].

Рассмотрим использование `freeGLUT` в Visual Studio 2008. Набор `freeglut` можно скачать в исходных кодах с сайта [4], но удобнее использовать откомпилированный вариант для Visual Studio, скачав его, например, с сайта [5] в виде архива `freeglut-MSVC-2.8.1-1.mp.zip`. Распакуем архив и поместим где-либо на диске, например, в папку `C:\OpenGL\freeglut` (рис.11.2). В этот архив включен файл `Readme.txt` с инструкцией, описывающей использование библиотеки.

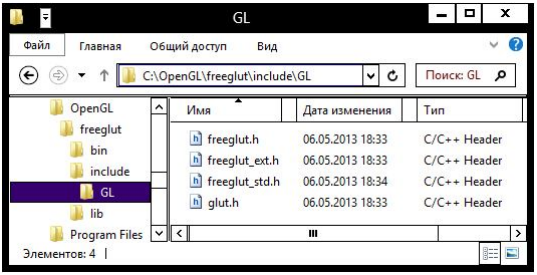

Рис. 11.2. Библиотека `freeglut`

Создадим в Visual Studio пустой проект по шаблону **`Консольное приложение Win32`** (рис. 11.3).

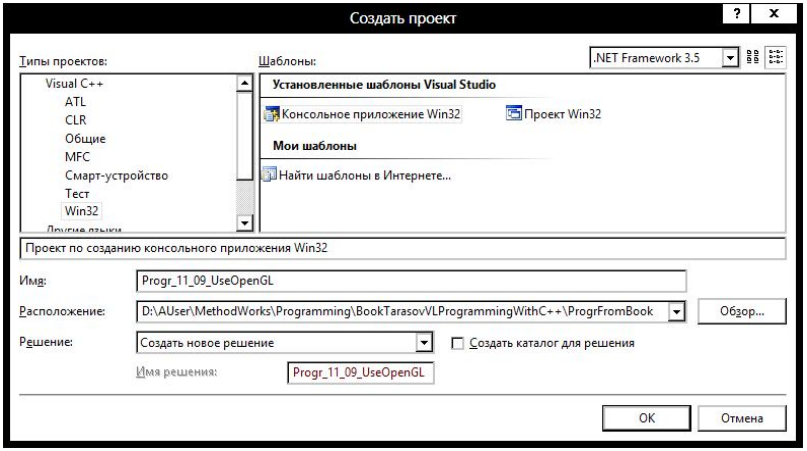

Рис. 11.3. Выбор шаблона создаваемого приложения

Для доступа к библиотеке **`freeglut`** сделаем следующие настройки.

Выполним команду **`Сервис`**, **`Параметры`**, раскроем секцию **`Проекты и решения`** и выберем ветку **`Каталоги VC++`**. Добавим в число каталогов, которые будут просматриваться при поиске включаемых файлов каталог `C:\OpenGL\freeglut\include\GL` (рис.11.4). Для этого следует нажать кнопку **`Создать строку`** с изображеием папки, затем кнопку ..., открывающую стандартный диалог открытия файла.

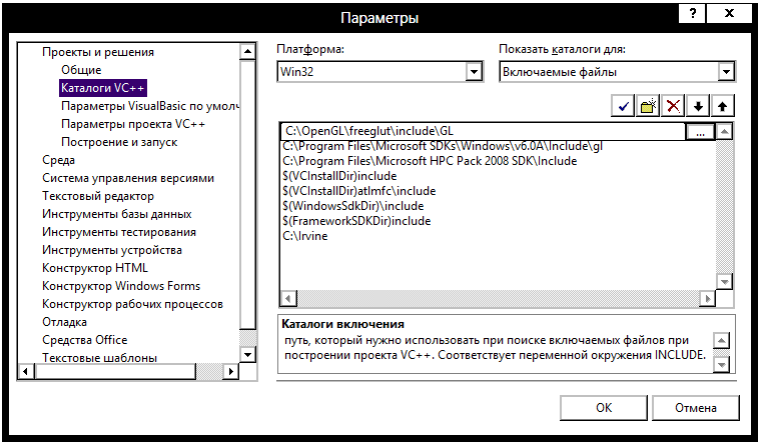

Рис. 11.4. Добавление каталога включаемых файлов библиотеки `freeglut`

Добавим также в раздел **`Файлы библиотек`** каталог `C:\OpenGL\freeglut\lib`, содержащий файл `freeglut.lib` (рис.11.5).

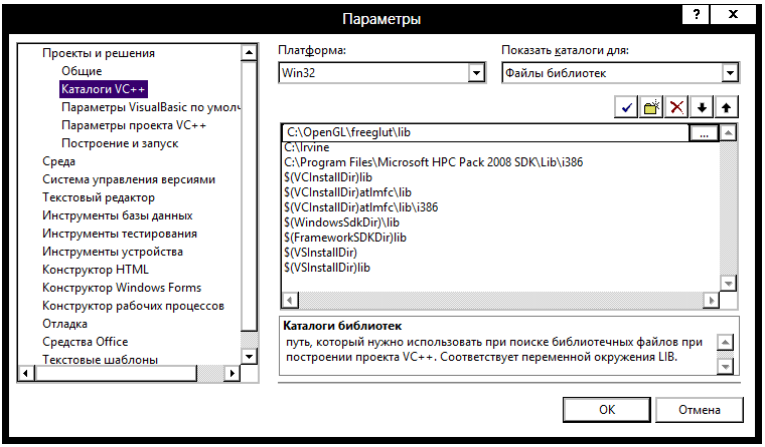

Рис. 11.5. Добавление каталога файлов библиотек `freeglut`

Программа 11.9. Схема использования OpenGL

Рассмотрим простую программу, использующую `freeglut`, которая рисует красный квадрат.

```
#include <stdlib.h>
#include <glut.h>
```

```
#include <iostream>
using namespace std;
#include <windows.h>

// keyboard: Функция для вызова при нажатии клавиши.
// Этой функции при вызове передаются:
// key - код символа нажатой клавиши,
// x, y - координаты курсора мыши в момент нажатия клавиши
void keyboard(unsigned char key, int x, int y)
{
    const int ESC = 27;              // Код клавиши Esc
    switch (key)
    {
        case ESC:
            exit(0);                 // Завершение программы
        break;
    }
}

// display: функция, вызываемая при необходимости перерисовать окно
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);    // Очистка буфера

    glColor3f(1.0f, 0.0f, 0.0f);     // Установка текущего цвета.
                            // Параметры - доли красного, зеленого и синего

    glBegin(GL_LINE_LOOP);           // Начало группы вершин
        glVertex2f(-0.5, -0.5);      // Левый нижний угол
        glVertex2f( 0.5, -0.5);      // Правый нижний угол
        glVertex2f( 0.5, 0.5);       // Правый верхний угол
        glVertex2f( -0.5, 0.5);      // Левый верхний угол
    glEnd();                         // Конец группы вершин
    glFlush();       // Функция glFlush() требует начать рисование
}

int main(int argc, char** argv)
{
    SetConsoleOutputCP(1251);
    glutInit(&argc, argv);           // Инициализация системы glut
    cout << "Создаю окно для рисования\n";
    glutCreateWindow("GLUT Test");   // Создание окна с заголовком
                                     // "GLUT test"
    glutKeyboardFunc(&keyboard);     // Регистрация функции keyboard,
                                     // для вызова при нажатии клавиши
    glutDisplayFunc(&display);       // Регистрация функции
                                     // для перерисовки окна
    glClearColor(1.0,1.0,1.0,1.0);   // Установка белого цвета фона
    glutMainLoop();                  // Цикл ожидания и обработки событий
    return 0;
}
```

Рассмотрим функцию `main()`. Ее работа начинается с вызова функции `glutInit()`, которая инициализирует систему `glut`. Ей следует

передать те аргументы, которые получает `main()` из своей командной строки.

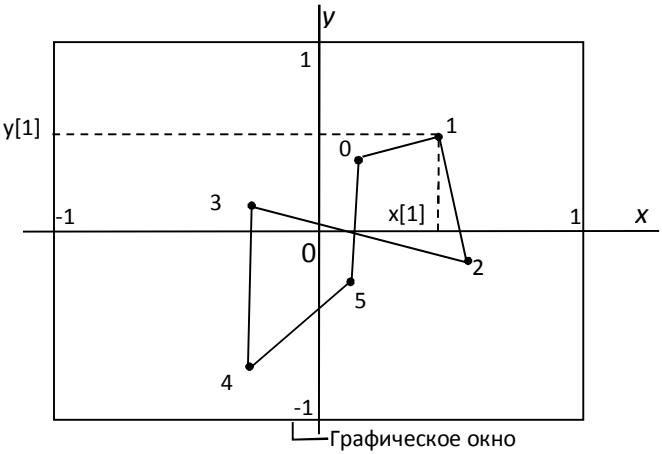

Рис. 11.6. Графическое окно, система координат и координаты точек

Функция `glutCreateWindow()` создает графическое окно для рисования с размерами и системой координат, задаваемыми по умолчанию. Схема графического окна и система координат иллюстрируются рис.11.6. Начало системы координат находится в центре окна, ось *x* направлена вправо, ось *y* вверх. Координаты изменяются в диапазоне от -1 до +1.

Функция `glutKeyboardFunc()` регистрирует функцию, которая будет вызываться при нажатии клавиш клавиатуры, для чего ей в качестве аргумента передается адрес функции `keyboard()`. Функция `keyboard(unsigned char key, int x, int y)` имеет три аргумента. В аргументе `key` передается код символа нажатой клавиши, `x` и `y` равны координатам курсора мыши в пикселях, отсчитываемых от левого верхнего угла графического окна. Функция `keyboard()` не реагирует на управляющие и функциональные клавиши. При нажатии клавиши `Esc` функция `keyboard()` вызывает `exit(0)`, которая завершает работу программы

Функция `glutDisplayFunc()` регистрирует функцию `display()`, которая вызывается для перерисовки графического окна при изменении его положения или размеров.

Функция `glutMainLoop()` создает цикл ожидания и обработки событий.

Рассмотрим теперь работу функции `display()`.

Функция `glClear()` очищает графическое окно от изображения.

Функция `glColor3f(1.0f, 0.0f, 0.0f)` устанавливает цвет рисования в соответствии с заданными долями красного (1.0), зеленого (0.0) и синего (0.0), то есть цвет рисования будет красный.

Функции `glBegin(GL_LINE_LOOP)` и `glEnd()` задают границы кода, между которыми можно задавать вершины рисунка. Параметр `GL_LINE_LOOP` указывает, что создаваемые вершины соединяются отрезками прямых, причем последняя вершина соединяется с первой. Таким образом, будет нарисована замкнутая ломаная линия.

Функция `glVertex2f()` задает вершину создаваемого изображения в плоскости графического окна.

После сделанных настроек компиляция и построение проекта, использующего библиотеку `freeglut`, будут выполняться успешно, но при запуске приложения обнаружится отсутствие динамической библиотеки `freeglut.dll` (рис. 11.7).

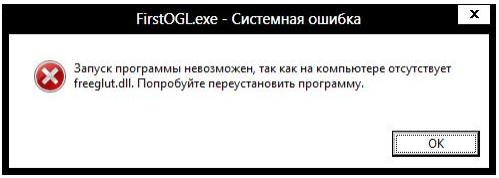

Рис. 11.7. Сообщение об отсутствии `freeglut.dll`

Файл `freeglut.dll` динамической библиотеки находится в папке, в которой установлена библиотека OpenGL (рис. 11.8).

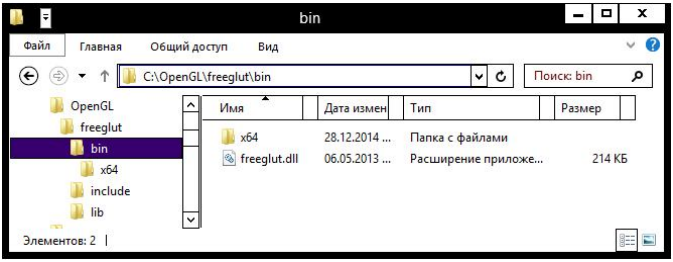

Рис. 11.8. Исходное расположение `freeglut.dll`

Самый простой способ дать доступ созданной программе к библиотеке `freeglut.dll` – это скопировать ее в папку, где расположен exe-файл приложения.

Если имеются права администратора, то можно скопировать `freeglut.dll` в системную папку (`C:\Windows\SysWOW64` для 64-разрядной

Windows), где расположен файл `opengl32.dll`. После этого программа запустится.

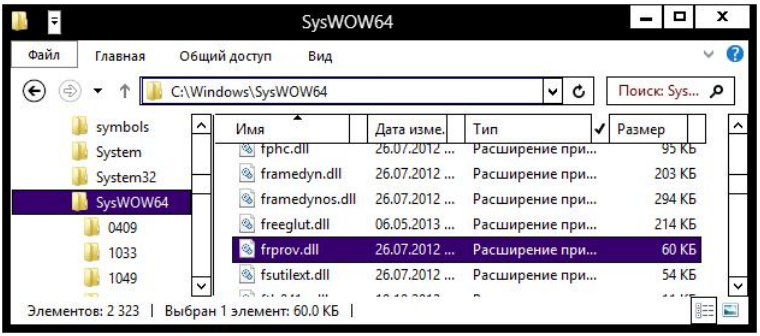

Рис. 11.9. Системная папка для `freeglut.dll`

Результат работы программы показан на рис.11.10. Приложение создает обычное консольное окно, через которое производится текстовый ввод и вывод, и отдельное графическое окно, в котором выполняется рисование.

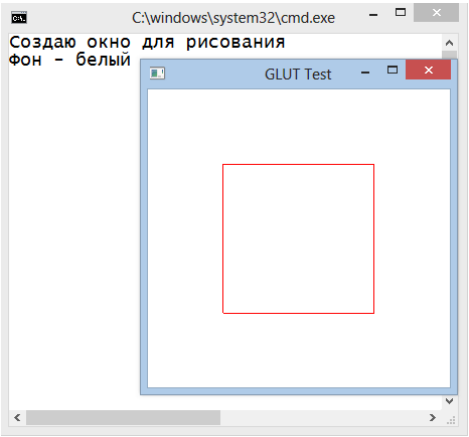

Рис. 11.10. Консольное и графическое окна программы

11.9. Деструкторы

В конструкторах при создании объектов могут захватываться некоторые ресурсы, например, выделяться память, открываться файлы и т. п. При уничтожении объектов ресурсы должны освобождаться. Для выполнения этих действий в состав класса включают специальные функции-члены – *деструкторы*.

Имя деструктора совпадает с именем класса с приставкой в виде знака ~ (тильда). Деструктор не имеет аргументов, и так же, как и конструктор, ничего не возвращает.

Деструкторы вызываются *неявно*, когда автоматическая переменная выходит из зоны видимости (заканчивается блок, в котором был создан объект), когда оператором `delete` удаляется объект, созданный в свободной памяти оператором `new`.

Программа 11.10. Деструктор в классе время дня

В классе `TimeDay` для времени дня деструктор не обязателен, так как при создании переменных этого класса никакие ресурсы не выделяются, кроме памяти под `hour` и `min`. Память под переменной типа `TimeDay`, занимаемая ею память освобождается встроенными средствами. Можно сказать, что при этом работает деструктор по умолчанию.

Включим, тем не менее, в состав класса `TimeDay` деструктор, который будет всего лишь сигнализировать о своей работе.

```
// Файл DestrTimeDay.h

#ifndef DestrTimeDayH
#define DestrTimeDayH

#include <iostream>
#include <iomanip>
#include <ctime>
#include <cstdlib>
using namespace std;

class TimeDay{                          // Время суток
    int hour;                           // Часы
    int min;                            // Минуты часа
public:
    TimeDay(int hh, int mm )            // Конструктор
    { hour = hh; min = mm;}
    ~TimeDay()                          // Деструктор
    {
        cout << "Работает деструктор для ";
        Print();
        cout << endl;
    }
    void Print()                        // Вывод времени
    {
    cout << setw(2) << setfill('0') << hour << ':'
```

```
            << setw(2) << setfill('0') << min << ' ';
}
};

#endif
```

Деструктор ~TimeDay() ничего существенного не делает, а только сигнализирует о своей работе.

В главной функции создаются три момента времени и выводятся.

```
// Файл DemoDestrTimeDay.cpp

#include "DestrTimeDay.h"

int main()
{
    setlocale(LC_ALL, "Russian");
    TimeDay Morning(6, 30);      // Создание трех
    TimeDay Dinner(12, 0);       // моментов
    TimeDay Evening(19, 45);     // времени
    cout << "Утро:  "; Morning.Print();
    cout << endl;
    cout << "День:  "; Dinner.Print();
    cout << endl;
    cout << "Вечер: "; Evening.Print();
    cout << endl;
    system("pause");
    return 0;
}       // При завершении программы освобождается память, занимавшаяся
        // Evening, Dinner и Morning с вызовом деструктора
```

Программа выдает:

```
Утро:  06:30
День:  12:00
Вечер: 19:45
Для продолжения нажмите любую клавишу . . .
Работает деструктор для 19:45
Работает деструктор для 12:00
Работает деструктор для 06:30
```

Данный пример подтверждает, что деструктор действительно вызывается неявно при уничтожении объектов класса, причем видно что деструктор вызывается в порядке, обратном порядку создания объектов, то есть в порядке, обратном порядку вызова конструкторов.

Программа 11.11. Многоугольники

В программе разработан класс `Plgn` для моделирования многоугольников на плоскости. Число вершин многоугольника может быть произвольным и задается при вызове конструктора, который выделяет память для координат вершин. В деструкторе память,

выделенная в конструкторе, освобождается. Объявление класса помещено в файл:

```
// Файл PoligonDestr.h

#ifndef POLIGON_DESTR_H
#define POLIGON_DESTR_H

# include <cstdlib>              // Доступ к rand()
# include <iostream>
# include <ctime>
using namespace std;

class Plgn                       // Класс многоугольников
{
    int nvert;                   // Число вершин
    float *x;                    // Указатель на массив абсцисс
    float *y;                    // Указатель на массив ординат
public:
    Plgn(int n = 3);             // Конструктор с аргументом по умолчанию
    ~Plgn();                     // Деструктор
    void PrnCoord();             // Вывод координат
};
#endif
```

В конструкторе создаются два динамических массива для двух координат вершин многоугольника и заполняются случайными числами из диапазона [-1, 1]. В деструкторе память, выделенная в конструкторе, освобождается.

```
//Файл PoligonDestr.cpp

#include "PoligonDestr.h"

Plgn::Plgn(int n)                // Конструктор
{
    nvert = n;
    x = new float[nvert];        // Выделение памяти под массив абсцисс
    y = new float[nvert];        // Выделение памяти под массив ординат

// Координатам присваиваем случайные значения от -1 до 1
    for(int i = 0; i < nvert; i++){
        x[i] = 2 * float(rand()) / RAND_MAX - 1;
        y[i] = 2 * float(rand()) / RAND_MAX - 1;
    }
}

Plgn::~Plgn()                    // Деструктор
{
    delete [] x;                 // Освобождение
    delete [] y;                 // памяти
}

void Plgn::PrnCoord()            // Вывод координат
{
```

```
    cout << "{ ";
    for(int i = 0; i < nvert; ++i)
        cout << "(" << x[i] << ", " << y[i] << ") ";
    cout << "}\n";
}
```

Координаты вершин многоугольника задаются в конструкторе случайными числами с использованием функции int rand(), которая генерирует псевдослучайное целое число из диапазона от 0 до RAND_MAX. Значение RAND_MAX доступно благодаря включению заголовка cstdlib, его значение 32767. Таким образом, случайные координаты вершин многоугольника будут находиться в диапазоне [-1, 1].

В данной программе деструктор обязателен, так как память, выделенную динамически с помощью оператора new, следует освобождать, вызывая явно оператор delete.

При использовании класса Plgn создается многоугольник по умолчанию (треугольник), и многоугольник с числом вершин, задаваемым при вводе.

```
// Файл UsePoligonDestr.cpp

#include "PoligonDestr.h"

int main()
{
    setlocale(LC_ALL, "Russian");
    srand(time(0));              // Инициализация датчика случайных чисел
    Plgn tr;                     // Многоугольник по умолчанию (труегольник)
    cout.precision(2);
    cout << "Создан треугольник:\n";
    tr.PrnCoord();

    cout << "Введите число вершин: ";
    int n;
    cin >> n;                    // Ввод числа вершин
    Plgn mng(n);                 // Многоугольник
    cout << "Создан многоугольник с " << n << " вершинами:\n";
    mng.PrnCoord();
    return 0;
}
```

Программы выводит:

```
Создан треугольник:
{ (0.99, -0.17) (0.0049, -0.48) (-0.58, -0.31) }
Введите число вершин: 5
Создан многоугольник с 5 вершинами:
{ (0.075, 0.92) (0.55, -0.18) (0.58, 0.45) (-0.21, -0.29) (0.79, -0.14) }
```

В данной программе при завершении main() автоматически вызывается деструктор для nmg и tr, который освободит память, выделенную в конструкторе.

Программа 11.12. Рисование многоугольников

Дополним программу 11.11 рисованием многоугольников с помощью библиотеки OpenGL.

```
// Файл Polygon.h

#ifndef POLIGON_H
#define POLIGON_H

# include <cstdlib>                  // Доступ к rand()
# include <iostream>
# include <ctime>
# define _USE_MATH_DEFINES
# include <cmath>
using namespace std;
# include <glut.h>
using namespace std;

class Plgn                           // Класс многоугольников
{
    int nvert;                       // Число вершин
    float *x;                        // Указатель на массив абсцисс
    float *y;                        // Указатель на массив ординат
public:
    Plgn(int n = 3);                 // Конструктор
    ~Plgn();                         // Деструктор
    void Show();                     // Функция рисования многоугольника
    void Move(float dx, float dy);   // Смещение многоугольника на dx, dy
    void Rotate(float angle);        // Поворот на угол angle градусов
};
#endif
```

Выпишем формулы, нужные для программирования поворота многоугольника вокруг центра масс. На рис.11.11 показан многоугольник и одна из его вершин M в исходном положении и после его поворота вокруг точки C на угол β. Новое положение вершины обозначено M_1.

В качестве центра масс можно взять точку C, координаты которой равны средним значениям координат вершин многоугольника:

$$x_C = \sum_{i=0}^{n-1} x_i \Big/ n, \quad y_C = \sum_{i=0}^{n-1} y_i \Big/ n.$$

Здесь n – число вершин многоугольника, x_i, y_i, i=0,…, n - 1 – координаты вершин. Данная формула действительно будет давать координаты

центра масс, если вся масса многоугольника сосредоточена в его вершинах и массы вершин одинаковы.

Как видно из рис.11.11, координаты точки *M* относительно центра масс *C* равны:

$$x_r = x - x_C = CM\cos\alpha, \quad y_r = y - y_C = CM\sin\alpha.$$

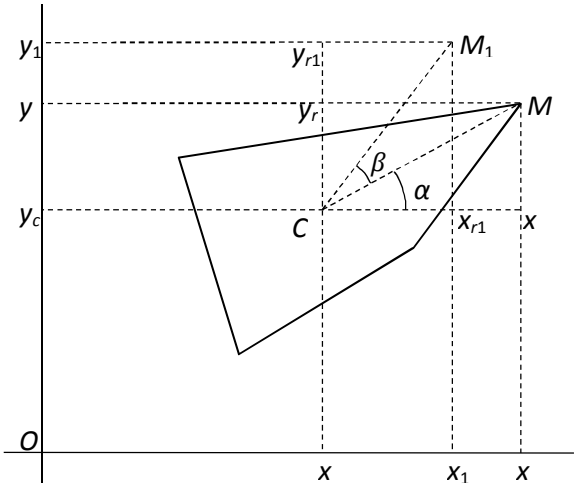

Рис. 11.11. Поворот многоугольника вокруг центра масс

Так как $CM_1 = CM$, то относительные координаты точки M_1 после поворота выразятся формулами:

$$x_{r1} = CM_1\cos(\alpha+\beta) = CM_1\cos\alpha\cos\beta - CM_1\sin\alpha\sin\beta = \\ = x_r\cos\beta - y_r\sin\beta,$$

$$y_{r1} = CM_1\sin(\alpha+\beta) = CM_1\sin\alpha\cos\beta + CM_1\cos\alpha\sin\beta = \\ = y_r\cos\beta + x_r\sin\beta.$$

Таким образом, новые абсолютные координаты точки *M* будут:

$$x_1 = x_C + x_{r1} = x_C + x_r\cos\beta - y_r\sin\beta,$$
$$y_1 = y_C + y_{r1} = y_C + y_r\cos\beta + x_r\sin\beta.$$

Эти формулы используются в функции вращения многоугольника.

Для рисования многоугольников будем использовать `glut` – оболочку над OpenGL, примененную в программе 11.9. Так как

координаты вершин многоугольника принадлежат диапазону [-1, 1], многоугольники не будут выходить за границы графического окна.

Реализация членов класса Plgn находится в файле Poligon.cpp.

```
// Файл Poligon.cpp

#include "Poligon.h"

Plgn::Plgn(int n)                    // Конструктор
{
    nvert = n;
    x = new float[nvert];            // Массив абсцисс
    y = new float[nvert];            // Массив ординат
    for(int i = 0; i < nvert; i++){
        x[i] = 2 * float(rand()) / RAND_MAX - 1;
        y[i] = 2 * float(rand()) / RAND_MAX - 1;
    }
}

Plgn::~Plgn()                        // Деструктор
{
    delete [] x;                     // Освобождение
    delete [] y;                     // памяти
}

void Plgn::Show()                    // Рисует многоугольник
{
    glClear(GL_COLOR_BUFFER_BIT);    // Очистка буфера
    glColor3f(1.0f, 0.0f, 0.0f);     // Установка текущего цвета.
    glBegin(GL_LINE_LOOP);           // Начало группы вершин
    for(int i = 0; i < nvert; i++)
        glVertex2f(x[i], y[i]);
    glEnd();                         // Конец группы вершин
    glFlush();                       // Функция glFlush()
}                                    // требует начать рисование

void Plgn::Move(float dx, float dy)  // Смещение многоугольника на dx по
{                                    // горизонтали и на dy по вертикали
    for(int i = 0; i < nvert; i++){  // Смещение
        x[i] += dx; y[i]+=dy;        // каждой вершины
    }
    Show();                          // Снова показываем многоугольник
}

void Plgn::Rotate(float angle)       // Функция поворота многоугольника
{                                    // на угол angle (задается в градусах)
    angle = angle * M_PI / 180.0f;   // Перевод угла в радианы
    float xC = 0, yC = 0;            // Координаты центра масс
    for(int i = 0; i < nvert; i++){
        xC += x[i]; yC += y[i];
    }
    xC /= nvert; yC  /= nvert;

    float xr, yr;           // Координаты вершины относительно центра масс
    for(int i = 0; i < nvert; i++){
```

```
        xr = x[i] - xC;              // Координаты вершины
        yr = y[i] - yC;              // до поворота
// Расчет координат вершин после поворота
        x[i] = xC + xr * cos(angle) - yr * sin(angle);
        y[i] = yC + xr * sin(angle) + yr * cos(angle);
    }
    Show();                          // Показываем многоугольник
}
```

Для рисования многоугольников с использование библиотеки glut надо написать функцию, которая бы вызывала метод Show() класса Plgn и зарегистрировать ее в системе. Регистрация выполняется функцией

```
void glutDisplayFunc(void (*pfunc)(void));
```

аргументом которой pfunc должен быть указатель на функцию, не имеющую аргументов. Для передачи в функцию рисования информации о многоугольнике, который надо изобразить, создадим глобальную переменную-указатель pp, который будет указывать на многоугольник, который следует нарисовать. Кроме этого, создадим два указателя, которые будут указывать на многоугольник и треугольник, созданные в main().

```
// Файл PoligonShow.cpp

#include "Poligon.h"

// Глобальные переменные
Plgn* pp;                // Указатель на изображаемый многоугольник
Plgn* ptriangle;         // Указатель на треугольник
Plgn* pplgn;             // Указатель на многоугольник
void display()
{
    pp->Show();
}

// Константы движения фигур
const float ANGLE = 5.0f;        // Угол поворота в градусах
const float STEP = 0.05f;        // Величина смещения
// Константы с кодами клавиш
const unsigned char
    ESC = 27,                    // Код клавиши Esc
    SPACE = 32;                  // Код клавиши "пробел"

// OrdinaryKeys: вызывается при нажатии обычных клавиш
void OrdinaryKeys(unsigned char key, int x, int y)
{
    switch(key){
        case ESC:
            exit(0);             // Завершение программы
        break;
        case '1':
            pp = pplgn;          // Выбираем многоугольник
```

```
            pp ->Show();
        break;
        case '2':                // Выбираем треугольник
            pp = ptriangle;
            pp ->Show();
        break;
        case SPACE:
            pp->Rotate(ANGLE);   // Вращение
        break;
    }
}

// SpecialKeys: вызывается при нажатии функциональных и
// управляющих клавиш

void SpecialKeys(int key, int x, int y)
{
    switch(key){
        case GLUT_KEY_LEFT :     // Нажата стрелка влево
            pp->Move(-STEP, 0);
        break;
        case GLUT_KEY_RIGHT :    // Нажата стрелка вправо
            pp->Move(STEP, 0);
        break;
        case GLUT_KEY_UP :       // Нажата стрелка вверх
            pp->Move(0, STEP);
        break;
        case GLUT_KEY_DOWN :     // Нажата стрелка вниз
            pp->Move(0, -STEP);
        break;
    }
}

void main(int argc, char* argv[])
{
    SetConsoleOutputCP(1251);
    int n;
    srand(time(0));              // Инициализация датчика случайных чисел
    cout << "Введите число вершин многоугольника: ";
    cin >> n;                    // Ввод числа вершин
    Plgn mnog(n);                // Многоугольник
    pp = pplgn = &mnog;          // Вначале работаем с многоугольником
    Plgn triangle;               // Многоугольник по умолчанию (треугольник)
    ptriangle = &triangle;
    cout << "Нажмите:\n"
        << "1 -   для работы с многоугольником\n"
        << "2 -   для работы с треугольником\n"
        << "Esc - для выхода\n";
    glutInit(&argc, argv);       // Инициализация системы glut
    glutCreateWindow("Многоугольники");  // Создание окна с заголовком
    glutKeyboardFunc(&OrdinaryKeys);     // Регистрация функции keyboard,
    glutSpecialFunc(&SpecialKeys);
    glutDisplayFunc(&display);           // Регистрация функции
    glClearColor(1.0,1.0,1.0,1.0);       // Установка белог цвета фона
```

```
    glutMainLoop();                // Цикл ожидания и обработки событий
}
```

Функция `OrdinaryKeys()` написана для обработки нажатия обычных клавиш. Ее действия: при нажатии <Esc> программа завершается; при нажатии клавиши <1> выбирается многоугольник; при нажатии клавиши <2> выбирается треугольник; при нажитии клавиши <пробел> фигура поворачивается на 5 градусов против часовой стрелки.

Перемещение многоугольника производится нажатием клавиш со стрелками. Для обработки нажатия этих клавиш выполняет функция:

```
void SpecialKeys(int key, int x, int y);
```

Нажатие алфавитно-цифровых клавиш приводит к формированию *однобайтового* кода соответствующего символа. При нажатии функциональных и управляющих клавиш формируется *двухбайтовый* код. Для этих кодов в файле `freeglut_std.h` определены константы вида `GLUT_KEY_<НазваниеКлавиши>`, например, `GLUT_KEY_LEFT`, `GLUT_KEY_RIGHT` и т.д. Для регистрации функции `SpecialKeys()` в `main()` вызывается функция:

```
void glutSpecialUpFunc(void (*pfunc)(int key,int x, int y));
```

В главной функции создается многоугольник с числом вершин, вводимым пользователем, и треугольник. Затем управление передается графической системе. С помощью функции `OrdinaryKeys()` выбирается одна из фигур и производится ее вращение с помощью клавиши «пробел». С помощью функции `SpecialKeys()` выполняется движение выбранной фигуры клавишами-стрелками.

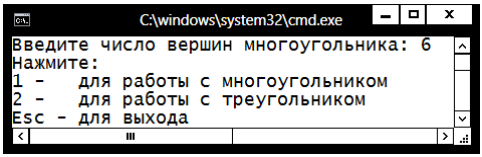

Рис. 11.12. Консольное окно программы

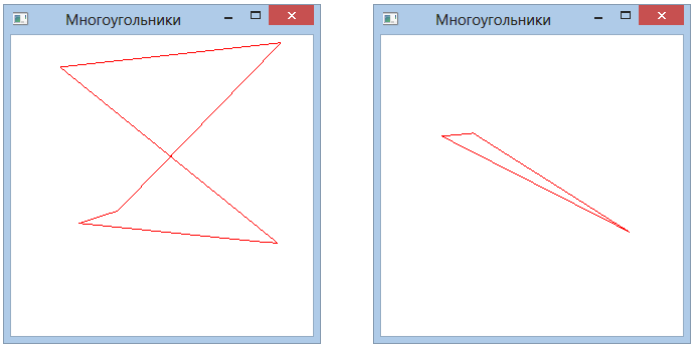

Рис. 11.13. Графические окна с многоугольником и треугольником

На рис.11.12 приведено исходное консольное окно, на рис.11.13 показано графическое окно, отображающее многоугольник и треугольник.

Литература к лекции 11

1. OpenGL Programming Guide. 8th Edition. The Official Guide to Learning OpenGL, Version 4.3/ Shreiner D, Sellers G, Kessenich J. M., Licea-KaneDave B.M.: [Электронный ресурс]. URL: http://it-ebooks.info/book/2138/. (Дата обращения: 05.01.2015)

2. OpenGL Red Book (русская версия): [Электронный ресурс]. URL: http://www3.msiu.ru/~kupri-ov/Books/RedBook_OpenGL.pdf. (Дата обращения: 05.01.2015)

3. GLUT – The OpenGL Utility Toolkit.: [Электронный ресурс]. URL: https://www.opengl.org/resources/libraries/glut/. (Дата обращения: 28.12.2014)

4. The Free OpenGL Utility Toolkit: [Электронный ресурс]. URL: http://freeglut.sourceforge.net/. (Дата обращения: 29.12.2014)

5. Transmission Zero. freeglut Windows Development Libraries: [Электронный ресурс]. URL: http://www.transmissionzero.co.uk/ software/ freeglut-devel/. (Дата обращения: 29.12.2014)