В.Л. Тарасов

Лекции по программированию на C++

Лекция 9

Структуры, перечисления

Для моделирования какого-либо предмета или явления требуется, как правило, несколько данных. Объединить в одно целое несколько данных позволяют структуры.

9.1. Структуры

Структура – это одна или несколько переменных, возможно, различных типов, которые сгруппированы в одно целое. Например, при создании электронной записной книжки каждый человек может описываться строкой символов с фамилией и числовым номером телефона.

В качестве простого примера рассмотрим моделирование времени суток. Пусть нас интересует время с точностью до минуты, например, при составлении расписания занятий. Для моделирования момента времени можно использовать следующую структуру:

```
struct TimeDay{      // Структура для моделирования времени суток
   int hour;         // Часы суток
   int min;          // Минуты часа
};
```

Здесь `struct` – это ключевое слово, с которого начинается описание структуры. Идентификатор `TimeDay` – это имя структуры. В фигурных скобках помещается список объявлений *членов* или *полей* структуры `hour` и `min`. Имена членов структуры должны различаться, но могут совпадать с именами других переменных программы.

Объявление структуры является отдельной инструкцией, создающей *новый тип данных*, поэтому оно завершается точкой с запятой. Именем нового типа данных является имя структуры, в данном примере это `TimeDay`. Структурный тип данных можно использовать так же, как встроенные типы. Например, чтобы создать переменные структурного типа, нужно записать:

```
TimeDay t1, t2, t3;          // Три переменные типа TimeDay
```

Эта запись аналогична объявлению переменных встроенных типов:

```
int a, b, c;
```

Переменные структурного типа могут копироваться, над ними можно выполнять операции присваивания, их можно передавать в функции в качестве аргумента и возвращать из функций в качестве результата.

Структурные переменные можно инициализировать при определении, например,

```
TimeDay StartCl = {7, 45};  // Начало занятий
```

Список инициализаторов заключается в фигурные скобки, в качестве инициализаторов можно использовать константы и константные выражения. Локальные переменные структурного типа, определенные внутри функции, можно инициализировать вызовами функций, которые возвращают результат структурного типа.

Доступ к члену структуры осуществляется оператором *точка* (.), которая разделяет имя переменной структурного типа и имя члена:

```
ИМЯ_СТРУКТУРЫ.ЧЛЕН
```

Например, напечатать время начала занятий можно инструкцией:

```
cout << StartCl.hour << ", " << StartCl.min;
```

9.2. Структуры и функции

Структуры можно передавать в функции в качестве аргумента; из функций можно возвращать структуру в качестве результата. Например, в программе 9.1 функция:

```
TimeDay MakeTimeDay(int h, int m)     // Создать структуру TimeDay
                                      // по часам h и минутам m
{
    TimeDay tmp;                      // Локальная структура внутри функции
    tmp.hour = h;                     // Заполнение членов
    tmp.min  = m;                     // структуры
    return tmp;                       // Возвращение структуры из функции
}
```

создает структуру по заданным значения часов h и минут m. Для этого внутри функции создается локальная структурная переменная tmp, поля которой заполняются заданными значениями. Инструкцией

```
return tmp;
```

значение созданной в функции структуры tmp возвращается в вызывающую программу.

Структуры можно передавать в качестве аргумента в функцию. Например, в программе 9.1 функция

```
void PrnTimeDay(const TimeDay& td);
```

получает в качестве аргумента ссылку на структуру td и выводит значения ее полей. Если передавать в функцию значение структуры, то на время работы функции будет создана копия структуры-аргумента, на что потребуется время и память. Так как структура может иметь большой объем, часто выгоднее передавать структуру в функции по ссылке, а не по значению. Если структура, передаваемая в функцию по ссылке не должна изменяться внутри функции, следует передавать ее по константной ссылке, как это сделано для функции PrnTimeDay().

В программе 9.1 функция

```
TimeDay DifTimeDay(const TimeDay& td1, const TimeDay& td2);
```

получает две структуры td1 и td2, содержащие два момента времени и определяет длительность промежутка времени между ними. Длительность промежутка возвращается из функции в виде структуры.

Программа 9.1. Структура для времени суток

В программе предлагается ввести текущее время и время окончания занятия, программа вычисляет и выводит, сколько времени осталось до окончания занятия. Код программы разместим в нескольких файлах. Объявление структуры и функций для работы с ней поместим в файл TimeDay.h.

```
// Файл TimeDay.h

#ifndef TimeDayH
#define TimeDayH

struct TimeDay{      // Структура для моделирования времени суток
   int hour;         // Часы суток
   int min;          // Минуты часа
};

TimeDay MakeTimeDay(int h, int m);            // Создать структуру TimeDay
void PrnTimeDay(const TimeDay&);              // Вывод структуры
TimeDay DifTimeDay(const TimeDay& td1, const TimeDay& td2);    // Промежуток
                                              // времени от td1 до td2

#endif
```

Реализацию функций для работы со структурой TimeDay поместим в файле TimeDay.cpp.

```
// Файл TimeDay.cpp
```

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

#include "TimeDay.h"

// MakeTimeDay: создание структуры

TimeDay MakeTimeDay(int h, int m)     // Создать структуру TimeDay
                                      // по часам h и минутам m
{
    TimeDay tmp;                      // Временная структура внутри функции
    tmp.hour = h;                     // Заполнение членов
    tmp.min  = m;                     // структуры
    return tmp;                       // Возвращение структуры из функции
}

// PrnTimeDay: вывод времени

void PrnTimeDay(const TimeDay& td)
{
    if(td.hour < 10)          // Если число часов однозначное,
        cout << '0';          // сначала выводится цифра нуль
    cout << td.hour << ':';   // Вывод часов
    if(td.min < 10)           // Если число минут однозначное,
        cout << '0';          // сначала выводится цифра нуль
    cout << td.min;           // Вывод минут
}

// DifTimeDay: возвращает промежуток времени от td1 до td2

TimeDay DifTimeDay(const TimeDay& td1, const TimeDay& td2)
{
    int m1 = td1.hour * 60 + td1.min;     // Число минут
                                          // от начала суток до td1
    int m2 = td2.hour * 60 + td2.min;     // Число минут
                                          // от начала суток до td2
    if(m2 < m1){
        cerr << "Первый момент времени позже второго\n";
        exit(1);
    }
    TimeDay tmp;
    tmp.hour = (m2 - m1) / 60;    // Число часов в промежутке времени
    tmp.min  = (m2 - m1) % 60;    // Число минут в промежутке времени
    return tmp;
}
```

Для тестирования разработанной структуры напишем программу, в которой создаются два момента времени и вычисляется длительность промежутка между ними.

```
// Файл TimeOfDay.cpp

#include <iostream>
#include <cstdlib>
#include <locale>
using namespace std;
```

```
#include "TimeDay.h"

int main()
{
    setlocale(LC_ALL, "Russian");
    cout << "Введите текущее время в часах и минутах:\n";
    int h, m;
    cin >> h >> m;
    TimeDay CurTime = MakeTimeDay(h, m);  // Создание и инициализация
                                          // структуры
    cout << "Введите время окончания занятий в часах и минутах:\n";
    cin >> h >> m;
    TimeDay EndClass = MakeTimeDay(h, m); // Еще структура

// Вычисляем время duration до конца занятия

    TimeDay duration = DifTimeDay(CurTime, EndClass);
    cout << "Занятие закончится через\n";
    PrnTimeDay(duration);                 // Вывод структуры
    system("pause");
    return 0;
}
```

В функции main() две структуры CurTime и EndClass инициализируются структурами, возвращаемыми функцией MakeTimeDay(). При инициализации и присваивании структур выполняется *почленное копирование*, то есть часы и минуты одной структуры копируются соответственно в часы и минуты другой сруктуры.

Далее приведен пример работы программы.

```
Введите текущее время в часах и минутах:
18 55
Введите время окончания занятий в часах и минутах:
21 0
Занятие закончится через
02:05
```

9.3. Вложенные структуры и вектора структур

Структуры могут быть вложенными друг в друга, например, для составления расписания занятий можно использовать структуру, включающую название предмета и время начала занятия:

```
struct ScheduleItem {        // Пункт расписания занятий
   string Subject;           // Название предмета
   TimeDay Start;            // Время начала занятия
};
```

Структура ScheduleItem имеет своими членами строку для названия учебного предмета и структуру TimeDay, для времени начала занятия.

Пусть объявлена структурная переменная:

```
ScheduleItem Progr;
```

Для заполнения полей структурной переменной Progr можно выполнить инструкции:

```
Progr.Subject = "Язык C++";
Progr.Start.hour = 9;
Progr.Start.min = 40;
```

Здесь в выражении Progr.Start.hour с помощью первого оператора «точка» (.) получается доступ к вложенной структуре Start типа TimeDay, а с помощью второго оператора «точка» реализуется доступ к полю Start. Следующая инструкция выводит значения полей структуры Progr:

```
cout << "Предмет " << Progr.Subject << ", начало: " << Progr.Start.hour <<
":" << Progr.Start.min;
```

Из структур можно создавать массивы и вектора. Например, для хранения информации о расписании из 4 занятий на некоторый учебный день можно создать вектор:

```
vector< ScheduleItem > Shed(4);
```

Программа 9.2. Расписание занятий

Применим структуру ScheduleItem для составления расписания занятий на один учебный день. За основу возьмем программу 9.1. В новый проект включим файлы TimeDay.h и TimeDay.cpp, благодаря чему в программе будет доступна структура TimeDay.

Объявление структуры ScheduleItem разместим в отдельном заголовочном файле Shedule.h.

```
// Файл Shedule.h

#ifndef ScheduleH
#define ScheduleH

#include <string>
using namespace std;

#include "..\\Progr_09_01_TimeOfDay\\TimeDay.h"

struct ScheduleItem {        // Пункт расписания занятий
    string Subject;          // Название предмета
    TimeDay Start;           // Время начала занятия
};

// Создать пункт расписания
```

```
ScheduleItem MakeSchaduleItem(string sub, const TimeDay& tm);

#endif
```

При вставке заголовочного файла TimeDay.h указано его полное имя, включая имя папки (каталога): "..\\Progr_09_01_TimeOfDay\\TimeDay.h". В операционной системе в качестве разделителя частей составного имени используется обратная наклонная черта, которая в программах на C++ изображается как '\\'. Две точки ".." означают корневой каталог для текущего каталога, в котором находится файл Shedule.h. Затем указан каталог, где расположен включаемый файл и сам этот файл TimeDay.h.

В файле Schedule.cpp разместим определение функции MakeSchaduleItem().

```
// Файл Schedule.cpp

#include "Schedule.h"

// Создать пункт расписания
ScheduleItem MakeSchaduleItem(string sub, const TimeDay& tm)
{
    ScheduleItem tmp;
    tmp.Start = tm;
    tmp.Subject = sub;
    return tmp;
}
```

В главной функции создается вектор из пунктов расписания размера, вводимого с клавиатуры, и заполняется вводимыми данными. Затем расписание выводится.

```
// Файл MakeShedule.cpp

#include <iostream>
#include <cstdlib>
#include <locale>
#include <vector>
using namespace std;

#include "Schedule.h"

int main()
{
    setlocale(LC_ALL, "Russian");
    cout << "Делаем расписание на завтра:\n";
    cout << "Введите число занятий: ";
    int nless;                          // Число занятий
    do{
        cin >> nless;
        if(nless < 0)
            cerr << "Вы ошиблись, повторите: ";
    }while(nless < 0);
    vector<ScheduleItem> shed(nless);     // shed - вектор, содержащий
```

```
                        // названия предметов и время начала занятий
    cout << "Введите названия " << nless
              << " предметов и время начала\n";
    cout << "Предмет  Начало\n";
    for(int i = 0; i < nless; ++i){
        cin >> shed[i].Subject;
        cin >> shed[i].Start.hour;
        cin >> shed[i].Start.min;
    }
    cout << "Ваше расписание\n";
    cout << "Предмет\tЧасы\tМинуты\n";
    setlocale(LC_ALL, ".866");
    for(int i = 0; i < nless; ++i){
        cout << shed[i].Subject << "\t";
        cout << shed[i].Start.hour << "\t";
        cout << shed[i].Start.min << endl;
    }
    system("pause");
    return 0;
}
```

Программа выводит:

```
Делаем расписание на завтра:
Введите число занятий: 3
Введите названия 3 предметов и время начала
Предмет  Начало
Алгебра 8 00
Физика  9 40
C++    11 20
Ваше расписание
Предмет Часы    Минуты
Алгебра 8       0
Физика  9       40
C++     11      20
```

В инструкциях вида

```
cout << shed[i].Subject << "\t";
```

при выводе символов табуляции (\t) курсор смещается в очередную позицию табуляции, благодаря чему можно сформировать при выводе ровные столбцы.

9.4. Перечисления

Перечисление – это список целых констант, создаваемый с помощью ключевого слова enum, например:

```
enum Boolean {NO, YES};
```

Первая константа NO в перечислении enum имеет значение 0, у следующей константы YES значение 1. Можно явно указывать значения констант в перечислении, например:

```
enum escapes {BELL = '\a', BACKSPACE = '\b', TAB = '\t', NEWLINE = '\n',
              VTAB = '\v', RETURN = '\r'};
```

Если не все константы заданы явно, они продолжают прогрессию, начиная с последнего заданного значения, например,

```
enum months {JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT,
             NOV, DEC};
```

Здесь значение FEB есть 2, значение MAR есть 3 и т.д.

Имена констант в различных перечислениях должны отличаться друг от друга.

Значения констант в одном перечислении могут быть одинаковыми.

Константы, объявленные в перечислении, могут участвовать в любых выражениях, где допустимо вхождение целых типов.

Перечисление является *типом данных*, поэтому можно объявлять переменные типа перечисления. Компилятор не контролирует, входят ли значения, присваиваемые таким переменным, в их тип. В следующей программе использовано перечисление months.

Программа 9.3. Перечисление месяцев

```
// Файл Enumeration.cpp

enum months {JAN = 1, FEB, MAR, APR, MAY, JUN, JUL,
                      AUG, SEP, OCT, NOV, DEC};
#include <iostream>
using namespace std;

int main()
{
    months m, nextm, m1;          // Три переменные типа months
    m = MAR;                      // Использование константы из перечисления
    nextm = months (m + 1);       // Преобразование целых (m + 1)
    m1 = months (-1);             // и (-1) к типу month
    cout << "m = " << m << endl;
    cout << "nextm = " << nextm << endl;
    cout << "m1 = " << m1 << endl;
    cin.get();                    // Ждем нажатия Enter
    return 0;
}
```

Выражения m + 1 и -1 имеют целый тип, поэтому они преобразуются к типу months с помощью выражений months (m + 1) и months (-1).

На рис.9.1 показано окно локальных переменных при пошаговом выполнении программы. Видно, что отладчик показывает значения из диапазона перечисления как имена констант, а значения не из диапазона (-1) в числовом виде.

Рис. 9.1. Значения переменных типа перечисления

Программа выдает:

```
m = 3
nextm = 4
m1 = -1
```

9.5. Функции как члены структуры

В языке C++ в состав структур могут входить не только данные, но и *функции*. Объединение в структуре и данных и функций для их обработки называется *инкапсуляцией*. Достоинством инкапсуляции является полное описание модели предметной области, для которой разрабатывается программа, в одном месте, что упрощает работу с моделью. Функции, входящие в структуру, имеют непосредственный свободный доступ к данным, хранящимся в структуре.

Модифицируем структуру `TimeDay` для работы со временем суток, включив в нее функции.

Программа 9.4. График занятий

```
// Файл TimeDayFunc.h

#ifndef TimeDayFuncH
#define TimeDayFuncH

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

struct TimeDay{              // Структура для моделирования времени суток
    int hour;                    // Часы суток
    int min;                     // Минуты часа
```

```
    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
    TimeDay Difference(const TimeDay& dt);    // Длительность промежутка
                                              // времени
};
#endif
```

В состав структуры `TimeDay` кроме данных `hour` и `min`, предназначенных для хранения часов и минут, включены функции: `Set()`, `AddHour()`, `AddMin()`, `Print()` и `Difference()`. Функции, объявленные внутри структуры, называются *функциями-членами*.

Функция `Set()` *определена* непосредственно внутри структуры, то есть, написан блок с телом этой функции, остальные функции в структуре лишь *объявлены*, так как дан только их заголовок.

Пусть объявлены три переменных, имеющих тип `TimeDay`:

```
TimeDay td1, td2, dt;
```

Каждая из этих трех переменных хранит собственное значение часов `hour` и минут `min`. Для вызова функции-члена структуры используется оператор "точка". Например, чтобы переменные хранили определенные моменты времени, вызовем для них функцию `Set()`:

```
td1.Set(9, 30);
td2.Set(12, 50);
```

Функции, объявленные в структуре, следует где-то определить. При *определении* функции-члена структуры следует указывать имя структуры, разделяя имя структуры и имя функции оператором *разрешения области видимости* (`::`), как это сделано в приводимом далее коде.

```
// Файл TimeDayFunc.cpp

#include "TimeDayFunc.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 << ' ';
}

// Difference: возвращает длительность промежутка времени
TimeDay TimeDay::Difference(const TimeDay& td)
{
    int m1 = hour * 60 + min;          // Число минут от начала суток
                                       // до первого момента времени
    int m2 = td.hour * 60 + td.min;    // Число минут от начала суток
                                       // до td
    if(m2 < m1){                       // Момент времени dt был раньше
        int m = m1;                    // Обмен значений m1 и m2
        m1 = m2; m2 = m;
    }
    TimeDay tmp;                       // Длительность промежутка времени
    tmp.hour = (m2 - m1) / 60;         // Число часов в промежутке времени
    tmp.min  = (m2 - m1) % 60;         // Число минут в промежутке времени
    return tmp;
}
```

При реализации функции `Print()` использованы так называемые функции-манипуляторы объявленные в заголовке `iomanip`, которые управляют выводом. Вызов `setw(2)` устанавливает ширину поля вывода в 2 позиции, вызов `setfill('0')` назначает символ `'0'` как символ-заполнитель. Благодаря таким настройкам для часов и минут при выводе отводится по 2 позиции, а если значение однозначное, выводится сначала `0`.

Для объявленных выше переменных `td1`, `td2`, `dt` можно следующим образом применить функцию `Difference()`:

```
dt = td1.Difference(td2);
```

Здесь `Difference()` определит длительность промежутка времени от `td1` до `td2`, а результат будет присвоен структурной переменной `dt`.

В главной функции вводится время начала занятий в вузе, а затем выводятся время конца каждой пары и время начала следующей. Также вычисляется общая длительность учебного дня.

```
// Файл UseTimeDay.cpp

#include "TimeDayFunc.h"

int main()
{
    setlocale(LC_ALL, "Russian");
    int hpair;                         // Длительность пары в часах
    int mpair;                         // Длительность пары в минутах
    int interval;                      // Длительность перерыва в минутах
    int hbeg, mbeg;
    int n;                             // Количество пар
```

```
    cout << "Введите время начала первой пары: ";
    cin >> hbeg >> mbeg;
    TimeDay tstart;                  // Время начала занятий
    tstart.Set(hbeg, mbeg);          // Установка времени начала занятий
    TimeDay pair = tstart;           // Установка времени в структуре pair
    cout << "Введите длительность пары в часах и минутах: ";
    cin >> hpair >> mpair;
    cout <<  "Введите длительность перерыва в минутах: ";
    cin >> interval;
    cout << "Введите количество пар: ";
    cin>> n;
    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 duration = tstart.Difference(pair);   // Длительность учеб. дня
    cout << "Длительность учебного дня: ";
    duration.Print();
    TimeDay morning;                     // Время подъема
    morning.hour = 6;                    // Непосредственный доступ
    morning.min = 30;                    // к членам структуры
    cout << "\nПодъем утром в ";
    morning.Print();
    cout << endl;
    system("pause");
    return 0;
}
```

Программа выводит:

```
Введите время начала первой пары: 8 0
Введите длительность пары в часах и минутах: 1 30
Введите длительность перерыва в минутах: 10
Введите количество пар: 6
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
Длительность учебного дня: 10:00
Подъем утром в 06:30
```

Здесь инструкции

```
    morning.hour = 6;                    // Непосредственный доступ
    morning.min = 30;                    // к членам структуры
```

обращаются непосредственно к членам структуры.

9.6. Встроенные функции

Функции, *определенные* внутри структуры или класса (о классах речь пойдет ниже), являются *встроенными* или `inline`. В программе 9.4 такой встроенной функцией является функция `TimeDay::Set(int hh, int mm)`. В точках вызова обычных функций компилятор помещает команды копирования аргументов в локальные переменные, создаваемые на время работы функции, и команду передачи управления на код функции. В конце кода функции помещается команда перехода обратно в точку вызова. Сам код обычных функций существует в памяти в единственном экземпляре.

В отличие от обычных функций, полный код встроенных функций вставляется в каждую точку их вызова, благодаря чему не нужны команды передачи управления, что ведет к экономии времени при использовании встроенных функций, но может увеличить общий размер рабочей программы. Поэтому для ускорения работы программы можно делать встроенными небольшие функции, которые часто вызываются.

С помощью ключевого слова `inline` можно явно указывать, что функция является встроенной. Это использовано в следующей программе.

Программа 9.5. Использование встроенных функций

```
// Файл TimeDayInline.h

#ifndef TimeDayInlineH
#define TimeDayInlineH

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

struct TimeDay{                  // Структура для времени суток
    int hour;                    // Часы суток
    int min;                     // Минуты часа
    void Set(int hh, int mm)     // Установка времени
    { hour = hh; min = mm; }

    // Объявление встроенной функции
    inline void AddHour(int nh); // Добавить nh часов.

    void AddMin(int nm);         // Добавить nm минут
    void Print();                // Вывод времени
};

// AddHour - встроенная функция-член структуры TimeDay
inline void TimeDay::AddHour(int nh)
```

```
{ hour = (hour + nh) % 24; }

// FirstHalf: проверяет принадлежит ли момент времени td первой
// половине для. Встроенная функция не член структуры

inline bool FirstHalf(TimeDay& td)
{ return td.hour < 12; }

#endif
```

Сейчас мы имеем дело с тремя встроенными функциями. Это функция-член структуры `Set()`. Она является встроенной по умолчанию так ее тело определено непосредственно в структуре.

Встроенная функция-член структуры `AddHour()` объявлена внутри структуры с ключевым словом `inline`, а определена вне структуры.

Встроенными могут быть независимые от структуры функции. Таковой является `FirstHalf()`.

Встроенные функции должны быть определены в том файле, в котором они используется. Поэтому в данной программе определения встроенных функций помещены в загловочный файл, который вставляется во все другие файлы, где будут использованы встроенные функции.

В следующем файле определяются остальные функции-члены структуры `TimeDay`.

```
// Файл TimeDayInline.cpp

#include "TimeDayInline.h"

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 << ' ';
}
```

Использование структуры со встороенными функциями демонстрирует следующая программа.

```
// Файл UseTimeDayInline.cpp

#include "TimeDayInline.h"
int main()
{
    setlocale(LC_ALL, "Russian");
    TimeDay work;                     // Время начала работы
    work.Set(8, 30);                  // Установка времени начала работы
```

```
    cout << "Начало работы: ";
    work.Print();
    if(FirstHalf(work) == true)       // Если еще нет 12 часов
        cout << endl << "Время обеда не наступило\n";
    work.AddHour(8);
    work.AddMin(50);
    cout << "Конец работы: ";
    work.Print(); cout << endl;
    system("pause");
    return 0;
}
```

Благодаря вставке файла `UseTimeDayInline.h` в файле `UseTimeDayInline.cpp` становятся известны определения встроенных функций `TimeDay::Set()`, `TimeDay::AddHour()` и `FirstHalf()`. Если поместить определения дружественных функций `TimeDay::AddHour()` и `FirstHalf()` в файл `UseTimeDayInline.cpp`, в котором находятся определения обычных функций, возникнет ошибка на этапе компоновки, так эти дружественные функции не будут найдены.

Программа выводит:

```
Начало работы: 08:30
Время обеда не наступило
Конец работы: 17:20
```

Компилятор может проигнорировать указание `inline`, когда, например, функция содержит условные операторы или циклы.