Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

А.А. Зарубин, Д.С. Краюхин, А.А. Шалыто

Система сбора данных на метеорологической станции (пример из книги Г. Буча)

Программирование с явным выделением состояний

Проектная документация

Проект создан в рамках движения за открытую проектную документацию
http://is.ifmo.ru

Санкт-Петербург
2005

Содержание

1. Введение

Несмотря на широкое распространение различных методологий проектирования, в литературе можно встретить лишь единичные экземпляры реализованных и документированных проектов. Данная работа имеет своей целью продемонстрировать процесс создания программного продукта от начала до конца, уделяя основное внимание его проектированию.

В качестве задачи был взят пример из восьмой главы книги Г. Буча [1] — система сбора данных на метеорологической станции. Анализ примера позволил сделать вывод, что Г. Буч не ставил своей целью реализацию проекта, так как при такой цели его текст лишился бы налета поверхностности и обилия недочетов и противоречий. В настоящей работе пройден путь проектирования и реализации, который «набросал» Г. Буч.

Стоит также заметить, что неудовлетворенность методами Г. Буча проявлялась и ранее, примером тому может служить работа [2], в которой управление кофеваркой было реализовано в трех вариантах: на основе метода Буча и SWITCH-технологии с одним и тремя автоматами. При этом авторами работы [2] был сделан вывод о том, что применение автоматного подхода является более естественным.

Данная работа относится к группе проектов, направленных на совершенствование классических примеров. В работе [3] Б. Страуструп реализует калькулятор, не выполняя его проектирование, а в работе [4] Д. Кнут так же моделирует лифт. Эти недостатки были устранены в работах [5, 6] соответственно.

У Г. Буча другая крайность — поверхностное проектирование и отсутствие реализации. В тех случаях, когда у него «зачатки» реализации появляются, проектирование и реализация перемешиваются. Это усугубляется его высказыванием, приведенным в работе [1], о том что «документация никогда не должна ставиться во главу угла при разработке».

В инженерной практике это не так. Более того, с этим не все согласны. Так, Э. Дейкстра пишет: «Глубоко ошибается тот, кто думает, что изделиями программистов являются программы, которые он пишет. Программист обязан создавать заслуживающие доверия решения и представлять их в форме убедительных доводов, а текст написанной программы является лишь сопроводительным материалом, к которому эти доказательства применимы». Все это привело к организации «Движения за открытую проектную документацию» [7], в рамках которого выполняется настоящий проект.

При выполнении проекта используется две технологии: предложенный Г. Бучем объектно-ориентированный анализ и автоматный подход, предложенный в работе [7], заменяющий использование диаграмм *statechart* на графы переходов. Применение автоматного подхода позволяет использовать не только такое понятие как «вложенное состояние», но и понятие «вложенный автомат», что обеспечивает возможность более компактного описания сложных алгоритмов. Компактность описания достигается также за счет применения в графах переходов не смысловых, а символьных обозначений.

Обратим внимание, что используемый автоматный подход отличается от классической теории автоматов [8] и подхода, основанного на диаграммах *statechart*.

В заключение раздела отметим, что после того как настоящая работа была практически завершена, на русском языке появилась работа [9], в которой также рассмотрен данный пример в обновленной форме, предназначенный для третьего издания книги Г. Буча. Однако, проектирование и реализация этого примера, приведенные в работе [9], не оставляют впечатления законченного проекта, особенно учитывая тот факт, что единственная диаграмма *statechart* из работы [1] здесь и вовсе исчезла, и понять реализованное поведение практически невозможно.

Авторы в настоящей работе решили поставить точку в проблеме проектирования и реализации этого примера, и довели его до конца, разработав не только исполняемый код, но и проектную документацию,

Проект реализован на языке *C++*. Автоматы в работе изображены с помощью программного продукта «Microsoft Visio».

2. Постановка задачи

Требования и ограничения перенесены без изменений из работы [1]. Относительно ограничений авторы считают необходимым привести некоторые комментарии.

2.1. Требования

Система должна обеспечивать автоматический мониторинг следующих первичных погодных параметров:

- скорость и направление ветра;
- температура;
- барометрическое давление;
- влажность воздуха.

Система также должна вычислять некоторые производные параметры, в число которых входят:

- относительное изменение температуры;
- относительное изменение барометрического давления.

В системе должна быть предусмотрена возможность определения текущего времени и даты, которые будут использоваться при генерации сообщении о максимальных и минимальных значениях первичных параметров за последние 24 ч.

Система должна обеспечивать постоянный вывод на дисплей текущих значений шести первичных и производных параметров, а также текущее время и дату. Пользователь должен иметь возможность увидеть максимальные и минимальные значения любого из первичных параметров за 24 ч, сопровождаемые информацией о времени произведения соответствующего замера.

Система должна также позволять пользователю проводить калибровку датчиков по известным опорным значениям, а также устанавливать текущее время и дату.

2.2. Ограничения

На рис.1 приведена структура моделируемой системы по Г.Бучу.

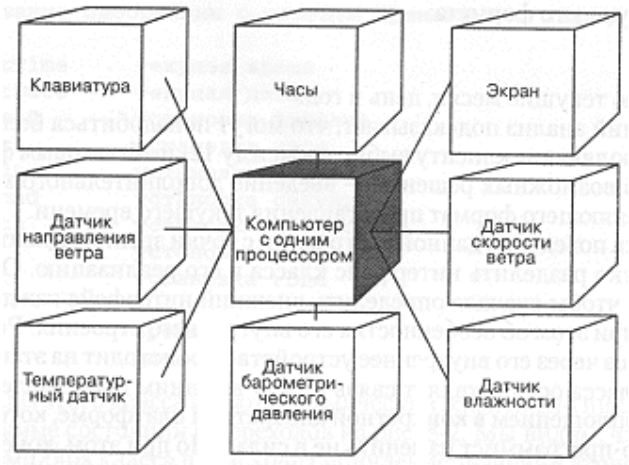

Рис. 1. Аппаратное обеспечение системы мониторинга погоды

Будем считать, что:

- используется компьютер с одним процессором;
- системные время и дата поддерживаются встроенными часами, а соответствующие значения отображаются в оперативную память;
- температура, барометрическое давление и влажность определяются встроенными контроллерами, которые соединены с соответствующими датчиками; показания контроллеров также отображены в оперативную память;
- направление ветра измеряется с помощью флюгера с точностью до одного из 16 направлений; скорость ветра определяется анемометром со счетчиком оборотов;
- ввод команд пользователем осуществляется с помощью клавишной панели (похожей на телефонную), сигналы которой обрабатываются с помощью встроенной платы. Эта же плата обеспечивает звуковой сигнал после каждого нажатия клавиши. Последняя команда пользователя сохраняется в памяти;
- экраном служит обычный жидкокристаллический дисплей. Встроенный контроллер дисплея обеспечивает вывод на экран небольшого набора графических примитивов: линий и дуг, закрашенных областей и текста;
- встроенный таймер посылает прерывания через каждую 1/60 с.

2.3. Замечания

Архитектура выполняемого проекта претерпела небольшие изменения по сравнению с архитектурой, представленной на рис.1, в части ее упрощения и в окончательном виде выглядит так, как показано на рис. 2.

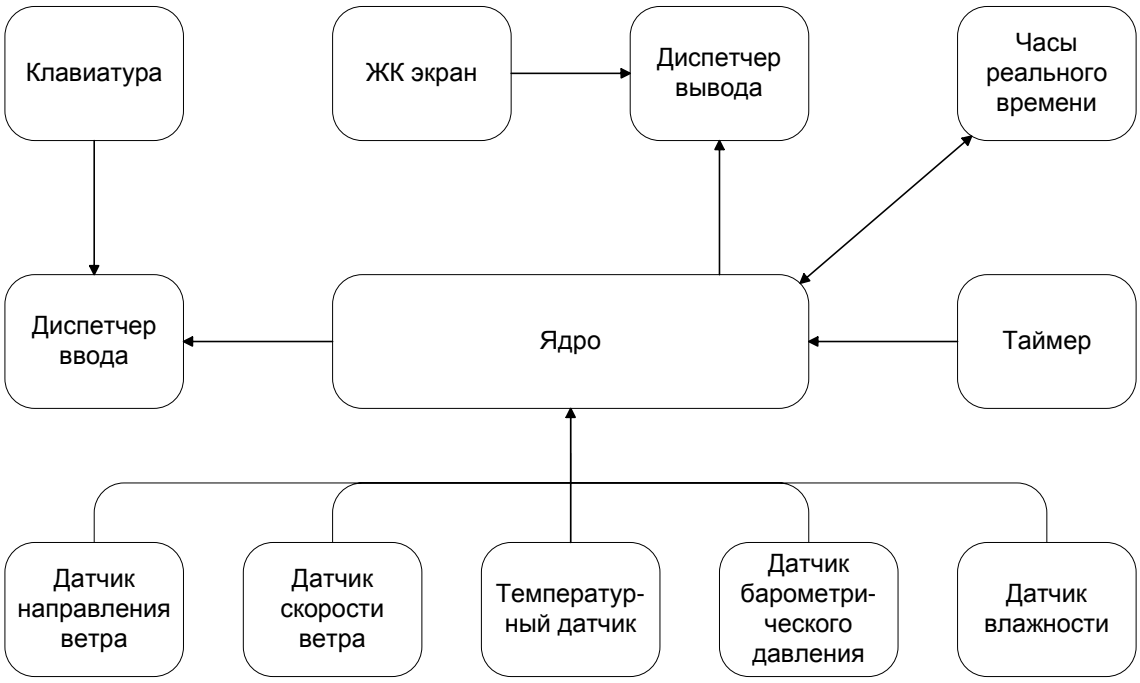

Рис. 2. Архитектура системы

В разд. 7 приведена диаграмма классов, каждый из которых соответствует одному из блоков архитектуры системы.

Система реализована в виде приложения *Windows*. Поэтому такие «аппаратные» компоненты системы, как жидкокристаллический (ЖК) экран и клавиатура превратились в два окна, внешний вид которых представлен на рис. 3, 4.

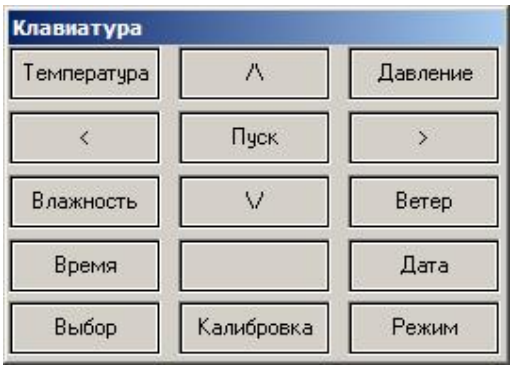

Рис. 3. Окно клавиатуры

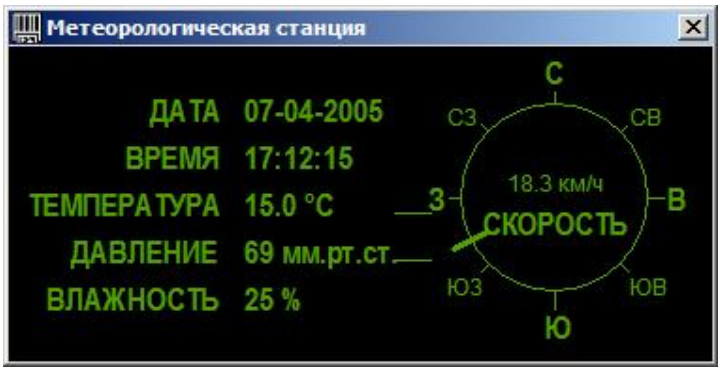

Рис. 4. Окно экрана

3. Анализ

Г. Буч в работе [1] под предлогом необходимости уделить внимание программной части находит применение своему объектно-ориентированному подходу. Конечно, удобно сказать, что ЖК экрана нет и начать выдумывать интерфейс класса для его программной реализации.

На практике большинство реальных информационно-управляющих систем приходится программировать на языке *C*, так как для микроконтроллеров не известны трансляторы с языка *C++*. При этом необходимо отметить, что, во-первых, в языке *C* объекты отсутствуют. Во-вторых, имея в своем распоряжении клавиатуру, экран, процессор, плату с часами реального времени и датчики, с которыми можно общаться по стандартному интерфейсу, Г. Буч был бы вынужден взяться за программирование поведения системы, которое он описал одной картинкой, названной им диаграммой, и парой абзацев текста…

Он не предполагал реализовывать систему, а для иллюстрации объектно-ориентированного подхода использовал язык *C++*. Поэтому и в настоящей работе применяется этот язык, но пример доводится до работающей модели на ПК.

Приводимые ниже описания взяты из работы [1], однако они структурированы и отделены от «попыток» реализации, которая выполнена позднее, так как у Г. Буча по ходу выполнения анализа вдруг появляются куски кода на языке *C++*.

3.1. Время и дата

Имеет смысл создать простую абстракцию для определения текущего времени и даты. Для этого необходимо провести небольшой анализ, в процессе которого придется рассмотреть ее роли и обязанности.[1] В частности, можно прийти к решению, что данная абстракция ответственна за предоставление информации о текущем времени в часах, минутах и секундах, а также о дате (текущий день, месяц и год). В результате анализа также можно сделать вывод о том, что среди обязанностей класса необходимо выделить две услуги: получение текущего времени и текущей даты.

Дальнейший анализ подсказывает, что могут понадобиться более полные абстракции, позволяющие клиенту выбирать между 12- и 24-часовым форматом времени. Одно из возможных решений — введение дополнительного модификатора, изменяющего формат представления текущего времени.

3.2. Температурный датчик

Абстракция «температурный датчик» служит аналогом аппаратного температурного датчика системы и занимается поддержанием информации о текущей температуре. Изолированный анализ ее поведения дает в первом приближении следующий результат: абстракция «температурный датчик» должна уметь выполнять операции «текущая температура», «установка минимальной температуры» и «установка максимальной температуры».

Название операции «текущая температура» говорит само за себя. Назначение двух других операций (установка минимальной и максимальной температур) прямо определяется требованием к системе — необходимостью проведения калибровки датчиков. Сигнал от каждого датчика — это число с фиксированной точкой из некоторого рабочего диапазона, граничные значения которого должны быть заданы. Промежуточные значения температуры вычисляются простой линейной интерполяцией между этими двумя точками, как показано на рис. 5.

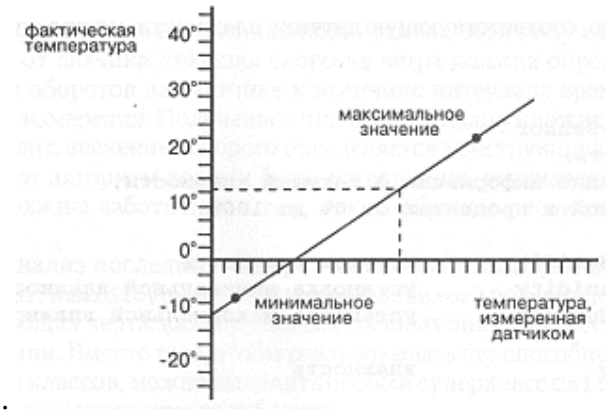

Рис. 5. Калибровка температурного датчика

В целях обеспечения возможности повторного использования абстракции она выделяется отдельно. На самом деле количество температурных датчиков не должно

[1] На самом деле, прежде чем придумывать самим, полезно порыться и постараться найти там что-нибудь похожее. Абстракция времени и даты — хороший кандидат на повторное использование, и скорее всего кто-нибудь уже разработал и отладил подобную абстракцию. Однако лучше предположить, что такой абстракции в готовом виде не нашлось.

влиять на архитектуру системы, и с выделением абстракции «температурный датчик» открывается возможность ее использования в других программах подобного типа.

3.3. Датчик барометрического давления

Абстракция для датчика барометрического давления может выглядеть следующим образом: она получает информацию о текущем барометрическом давлении и выполняет операции: «возвращение значения текущего давления», «установка минимального давления» и «установка максимального давления».

3.4. Тренд

При более подробном рассмотрении требований к системе выясняется, что упущена одна важная характеристика поведения перечисленных выше абстракций - как изменяются температура и барометрическое давление (его относительное изменение, тренд). На этапе анализа, обратим основное внимание на природу этого поведения и, самое важное, выясним, какая абстракция должна отвечать за него.

И для температурного датчика, и для датчика бараметрического давления тренд может быть определен как вещественное число, изменяющееся в диапазоне от минус единицы до единицы и представляющее собой наклон графика изменений температуры и давления на некотором интервале времени.[2] Таким образом, к описанию двух вышеупомянутых абстракций можно добавить ответственность за определение тренда и соответствующую ей операцию «тренд».

Вообще говоря, подобная схема не является единственно возможной. Тем не менее, было решено передать ответственность за определение тренда датчикам. Можно было бы создать внешнюю по отношению к датчикам абстракцию, ответственную за периодический опрос датчиков и вычисления трендов, но такой вариант был отвергнут, так как он неоправданно усложнил бы систему. Первоначальное описание абстракций температурного датчика и датчика давления уже подразумевало возможность вычисления тренда «своими силами», а выявив общность, была получена простая и связная система абстракций.

3.5. Датчик влажности

Абстракцию, соответствующую датчику влажности, можно определить следующим образом: она получает и формирует информацию о текущей влажности, выраженной в процентах от 0 до 100%, и выполняет операции: «возвращение значения текущей влажности», «установка минимальной влажности» и «установка максимальной влажности».

Задача определения тренда влажности не ставится.

[2] Значение ноль показывает, что температура или давление стабильно. Значение 0,1 указывает на небольшой рост, значение –0,3 соответствует резкому уменьшению параметра. Значения, близкие к –1 и 1, отражают природный катаклизм, выходящий за рамки тех сценариев, в которых система должна исправно работать. На роль функции тренда вполне подходит синус угла наклона касательной в данной временной точке.

3.6. Максимальное и минимальное значения

Требования к системе подразумевают наличие общего поведения для всех трех вышеперечисленных абстракций. В частности, необходимо обеспечить показ максимального и минимального значений каждого параметра за 24 ч. Эту обязанность можно отразить в следующем описании, общем для всех трех абстракций: новая абстракция должна заниматься генерацией сообщений о максимальных и минимальных значениях параметров за 24 ч — поддерживать операции: «возвращение максимального и минимального значений», «возвращение времен, соответствующих максимальному и минимальному значениям».

3.7. Датчик скорости ветра

Абстракция для датчика скорости ветра может выглядеть следующим образом: она получает информацию о текущей скорости ветра и выполняет операции: «возвращение значения текущей скорости», «установка минимальной и максимальной скоростей».

3.8. Калибровка

Краткий анализ последних четырех абстракций системы («температурный датчик», «датчик барометрического давления», «датчик влажности» и «датчик скорости ветра») показывает, что у них имеется еще одна общая черта: калибровка измеренных значений посредством линейной интерполяции. Эту способность не требуется реализовывать по отдельности для каждой абстракции. Необходимо создать еще одну абстракцию, обеспечивающую линейную интерполяцию значений, лежащих в известном интервале и выполняющие операции: «возвращение текущее значение», «установка минимального и максимального значений».

3.9. Датчик определения направления ветра

Последний рассматриваемый датчик — датчик определения направления ветра — несколько отличается от всех остальных, так как он не нуждается ни в калибровке, ни в вычислении минимальных и максимальных значений. Можно определить данную абстракцию следующим образом: она должна будет получать информацию о текущем направлении ветра, указываемом как точка на розе ветров, и потребуется операция «возвращение направления».

3.10. Ввод с клавиатуры

Абстракция ввода информации с клавиатуры имеет следующий вид: необходимо получать и хранить информацию о коде последней клавиши, нажатой на клавиатуре. Следовательно, должна существовать операция «возвращение значения о нажатии последней клавиши».

Заметим, что эта абстракция не знает о назначении той или иной клавиши: данный экземпляр несет информацию лишь о том, какая клавиша была нажата.

3.11. Вывод на экран

Следующая абстракция — «дисплей», предназначенный для того, чтобы обеспечить определенную независимость программной системы от аппаратной части, на которой она будет работать. Для рабочих станций и персональных компьютеров существует целый ряд стандартов (хотя зачастую и конфликтующих между собой) графического интерфейса, таких, например, как *Motif* или *Microsoft Windows*. К сожалению, для микроконтроллеров нет общепризнанных стандартов, поэтому анализ задачи приводит к мысли о том, что для ее решения необходимо создать прототипы и затем определить основные требования к интерфейсу пользователя.

На рис. 4 приведен один из подобных прототипов. Он содержит все основные графические элементы. Будем выводить на экран текст (двух различных размеров и начертаний), окружности и линии различной толщины. Также следует заметить, что некоторые элементы изображения являются статическими (такие, как заголовок ТЕМПЕРАТУРА), а некоторые — динамическими (например, направление ветра). И статические, и динамические элементы изображения генерируются программно. В итоге упрощается аппаратная часть (не надо заказывать специальные жидкокристаллические дисплеи со встроенными статическими элементами), но несколько усложняется программное обеспечение.

Аналогично абстракции «клавиатура», абстракция «дисплей» не понимает, зачем она выводит тот или иной элемент на экран. Это дает возможность свободно оперировать ранее созданными абстракциями, однако требует наличия некоего внешнего агента, выполняющего функции посредника между датчиками и дисплеем.

4. Проектирование

> Способ управления сложными системами был известен еще в древности — *divide et impera* (разделяй и властвуй).
>
> Э. Дейкстра

> Цель проектирования — выявление ясной и относительно простой внутренней структуры, иногда называемой архитектурой ... Проект есть окончательный продукт процесса проектирования.
>
> Б. Страуструп

Этот раздел также берет свое начало из книги [1], но затем трансформируется в предлагаемый подход.

4.1. Таймер

Можно долго спорить, должна ли эта абстракция появиться в процессе анализа или в процессе проектирования. Авторы предпочли выбрать второй вариант, сочтя его более предпочтительным. На самом деле, можно поступать и так, и так, потому что эта абстракция расположена где-то на границе.

Сделаем упрощающее предположение о том, что таймер будет один на всю систему, и что системные прерывания будут осуществляться с периодичностью 60 раз в секунду.

Лучше, если детали реализации подобного таймера будут скрыты от остальных абстракций.

Как клиент взаимодействует с таймером? Сначала клиент передает таймеру функцию обратного вызова[3], а затем с периодичностью в 0,1 с таймер вызывает эту функцию. Тем самым клиент не заботится о том, как осуществляются прерывания, а таймер не знает, что при этом прерывании делать. Единственным требованием к клиенту должно быть ограничение на продолжительность выполнения функции обратного вызова — оно не должно превышать 0,1 с. В противном случае, таймер может пропустить событие.

Абстракция «таймер» осуществляющая прерывания, является активной абстракцией, она инициирует цепочку управляющих команд. Ее можно формализовать с помощью следующего описания: абстракция будет заниматься осуществлением прерываний и диспетчеризацией функции обратного вызова, предоставлять единственную операцию «установка функции обратного вызова».

4.2. Агент

Главной задачей системы является мониторинг основных измеряемых параметров. Одним из ограничений является невозможность обрабатывать информацию с частотой, превышающей 60 измерений в секунду. Однако наиболее интересные погодные параметры изменяются с гораздо меньшей скоростью. Дополнительный анализ показывает, что для своевременной регистрации изменений различных погодных параметров достаточно обеспечить следующие частоты снятия информации:

- каждые 0,1 с — направление ветра;
- каждые 0,5 с — скорость ветра;
- каждые 5 мин. — температура, барометрическое давление и влажность.

Ранее было принято решение о том, что классы датчиков не должны отвечать за организацию периодических измерений. Эта работа лежит в сфере ответственности внешнего агента, взаимодействующего с датчиками. Когда агент начинает обработку измерений, он последовательно опрашивает датчики. Однако при этом он может пропускать те датчики, для которых интервал опроса больше 0,1 с. При такой организации опроса, в отличие от схемы, в которой каждый датчик самостоятельно отвечает за измерение, обеспечивается более предсказуемое поведение системы, так как контроль за процессом считывания параметров сосредоточен в одном месте — в экземпляре агента.

Главная операция абстракции «агент» — «заме́р». Эта функция по очереди опрашивает каждый тип датчика и каждый датчик внутри типа. Она проверяет, пришло ли время считывать информацию с датчика, и если это время наступило, то определяет ссылку на датчик в коллекции, считывает его текущее значение и передает его менеджеру дисплея, ассоциированному с данным экземпляром абстракции «агент».

4.3. Диспетчер вывода

Теперь предстоит решить, кто должен отвечать за вывод информации на экран дисплея. Здесь возможны два варианта: можно передать ответственность за эти действия самим датчикам (подобная схема реализована в архитектурах, подобных «модель – вид – контроллер»), либо создать отдельную абстракцию для связи между датчиками и дисплеем. В данном случае был выбран второй вариант, так как тогда можно изолировать все проектные решения, касающиеся механизмов реализации вывода параметров на экран.

[3] Функциая обратного вызова – это функция, которая в отличие от прямого вызова, вызывается не самой программой, а внешним приложением для уведомления его о каком-либо событии.

В итоге к результатам анализа добавляется описание еще одной абстракции, которая занимается организацией отображения параметров на экране дисплея и выполняет операции «рисование статических элементов», «вывод времени», «вывод даты», «вывод температуры», «вывод влажности», «вывод давления», «вывод скорости ветра», «вывод направления ветра» и «вывод максимальных и минимальных значений».

Операция «отображения статических элементов» выводит на экран ту часть изображения, которая не изменяется в процессе работы системы, например, розу ветров для индикации направления ветра.

Отметим еще одно важное преимущество решения о выделении отдельной абстракции «диспетчер вывода». Задача локализации системы для различных стран предполагает изменение языка, на котором информация выводится на дисплей. Наличие отдельного класса, ответственного за вывод сообщений на экран, существенно облегчает процесс локализации, так как имена всех сообщений (например, ТЕМПЕРАТУРА, или СКОРОСТЬ) находятся в этом случае в ведении единственного класса. Они не разбросаны по множеству различных абстракций.

4.4. Вторичные параметры

Мониторинг вторичных параметров, в частности, трендов температуры и давления, можно обеспечить на основе протоколов уже приведенных ранее абстракций «температурный датчик» и «датчик барометрического давления».

4.5. Диспетчер ввода

Поведение системы описывается набором вложенных конечных автоматов. Поэтому разумно было бы управлять системой с помощью обособленной абстракции, задаваемой по следующим правилам: она предназначена для диспетчеризации команд пользователя с помощью операции «обработка сигналов от клавиатуры».

4.6. Архитектурный каркас

На этом этапе необходимо четко определить архитектуру проекта. Это даст фундамент, на основе которого будут строиться отдельные функциональные части системы.

Существует ряд архитектурных моделей для решения задач сбора и обработки данных и управления, но наиболее часто встречаются синхронизация автономных исполнителей и схема покадровой обработки.

В первом случае архитектура системы образована рядом относительно независимых объектов, каждый из которых выполняется как поток управления. Можно было бы, например, создать несколько новых датчиков, построенных с помощью более примитивных абстракций, каждый из которых отвечал бы за считывание информации с определенного датчика и за передачу ее центральному агенту, обрабатывающему всю информацию. Подобная архитектура имеет свои преимущества и является, пожалуй, единственной приемлемой моделью в случае проектирования распределенной системы, которая должна производить обработку большого числа параметров, поступающих с удаленных датчиков. Эта модель также позволяет эффективнее оптимизировать процесс сбора данных. Каждый объект-датчик может содержать в себе информацию о том, как

следует приспосабливаться к изменению окружающих условий — увеличивать или уменьшать частоту опроса, например.

Однако описанная архитектура оказывается не всегда приемлемой при создании жестких систем реального времени, где требуется обеспечить предсказуемость процесса обработки. Метеорологическую станцию нельзя отнести к таким системам, но для нее, тем не менее, требуется определенная степень предсказуемости и надежности. По этой причине для системы была выбрана модель покадровой обработки.

Процесс мониторинга осуществляется в данном случае как последовательность считывания, обработки и вывода на экран значений параметров через определенные промежутки времени. Каждый элемент такой последовательности называется *кадром.* Его, в свою очередь, можно разбить на ряд подкадров, соответствующих определенному функциональному поведению. Различные кадры могут нести информацию о различных параметрах. Направление ветра, например, необходимо измерять через каждые 10 кадров, а скорость ветра — через 30 кадров[4] Основное преимущество такой модели состоит в том, что появляется возможность более жестко контролировать последовательность действий системы по сбору и обработке информации.

Кроме того, вводится одна новая абстракция «сенсоры», которая служит для объединения в коллекцию всех объектов-датчиков. Поскольку, по крайней мере, два агента («агент» и «диспетчер ввода») в системе должны ассоциироваться с целой коллекцией датчиков, помещение их в один контейнерный класс позволяет рассматривать все датчики единым образом.

5. Автоматный подход

> Составление диаграмм — это еще не анализ и не проектирование. Диаграммы позволяют описать поведение системы (для анализа) или показать детали архитектуры (для проектирования).
> … четкая система обозначений позволяет автоматизировать большую часть утомительной проверки на полноту и правильность.
>
> Г. Буч

Безусловно, рано или поздно следовало приступить к важной стадии проектирования — описанию поведения системы. Г. Буч в своей книге [1] предпочитает использовать так называемые «сценарии».

Со слов Г. Буча, приведенных в эпиграфе, складывается впечатление, что он придерживается не лучшего мнения в отношении диаграмм. У авторов настоящей работы по этому вопросу другая точка зрения — удачно построенная система графов переходов поможет не только отразить результат проектирования, но и помочь в ходе их выполнения. В плане проверки на правильность можно согласиться с Г. Бучем: имея на руках граф переходов достаточно протестировать сами переходы, так как код программы изоморфен графу.

[4] Например, если кадры считываются через каждую 1/60 с, то 30 кадров занимают 0,5 с

5.1. Выделение состояний

Определив в рамках системы основные абстракции, продолжим анализ задачи и рассмотрим некоторые сценарии работы системы. Начнем с составления списка ситуаций. С точки зрения пользователя список будет выглядеть примерно следующим образом:

- мониторинг первичных измеряемых параметров (скорость и направление ветра, температура, барометрическое давление и влажность);
- мониторинг производных параметров: трендов температуры и барометрического давления;
- показ максимальных и минимальных значений выбранных параметров;
- установка времени и даты;
- калибровка выбранных датчиков;
- включение системы.

5.2. Сценарии

Далее рассмотрим различные сценарии взаимодействия пользователя и системы. Предоставление пользователю оптимальной последовательности действий для выполнения его задач является так же, как и проектирование графического интерфейса, в большой степени искусством.

Рассмотрим некоторые из возможных сценариев взаимодействия пользователя с системой.

5.2.1. Вывод на экран максимальных и минимальных значений выбранного параметра

Приведем сценарий, обеспечивающий вывод на экран макимального и минимального значеий

1. Пользователь нажимает клавишу *Выбор*.
2. Система выводит на экран сообщение *ВЫБОР*.
3. Пользователь нажимает одну из следующих клавиш: *Ветер*, *Температура*, *Давление* или *Влажность*; нажатие всех остальных клавиш (кроме клавиши *Пуск*) игнорируется.
4. Название выбранного параметра начинает мигать на экране.
5. Пользователь нажимает одну из клавиш /\ (*стрелка вверх*) или \/ (*стрелка вниз*), выбирая тем самым, какое значение — максимальное или минимальное — будет выведено на экран; нажатие всех остальных клавиш (кроме клавиши *Пуск*) игнорируется.
6. Система выводит на экран выбранное значение, а также время его замера.
7. Переход управления к пункту 3 или 5.

Замечание. Для прекращения работы в данном режиме пользователь нажимает клавишу *Пуск*. При этом экран дисплея возвращается в первоначальное состояние.

5.2.2. Установка времени и даты

Установка времени и даты производиться следующим образом.

1. Пользователь нажимает клавишу *Выбор*
2. Система выводит на экран сообщение *ВЫБОР*.
3. Пользователь нажимает одну из следующих клавиш: *Время* или *Дата*; нажатия всех остальных клавиш (кроме клавиши *Пуск* и клавиш, перечисленных в пункте 3 предыдущего сценария) игнорируются.
4. Название выбранного параметра, а также первое поле его значения (для времени — это час, для даты — месяц) начинают мигать на экране.
5. Пользователь нажимает одну из клавиш < (*стрелка влево*) или > (*стрелка вправо*) для перехода на другое поле; пользователь нажимает одну из клавиш /\ (*стрелка вверх*) или \/ (*стрелка вниз*) для увеличения или уменьшения значения выделенной величины.
6. Переход управления к пункту 3 или 5.

Замечание. Для прекращения работы в данном режиме пользователь нажимает клавишу *Пуск*. При этом экран дисплея возвращается в первоначальное состояние, и происходит переустановка времени и даты.

5.2.3. Калибровка датчика

Сценарий калибровки датчика производиться следующим образом.

1. Пользователь нажимает клавишу *Калибровка.*
2. Система выводит на экран сообщение КАЛИБРОВКА.
3. Пользователь нажимает одну из следующих клавиш: *Ветер*, *Температура*, *Давление* или *Влажность*; нажатия всех остальных клавиш (кроме клавиши *Пуск*) игнорируются.
4. Название выбранного параметра начинает мигать на экране.
5. Пользователь нажимает одну из клавиш /\ (*стрелка вверх*) или \/ (*стрелка вниз*), задавая тем самым, какое калибровочное значение, максимальное или минимальное, будет переопределяться.
6. Соответствующее калибровочное значение начинает мигать на экране.
7. Пользователь нажимает клавиши /\ (*стрелка вверх*) или \/ (*стрелка вниз*) для изменения значения выделенной величины.
8. Переход управления к пункту 3 или 5.

Замечание. Для прекращения работы в данном режиме пользователь нажимает клавишу *Пуск*. При этом экран дисплея возвращается в первоначальное состояние, и происходит перерасчет калибровочной функции.

На время калибровки все экземпляры агента должны прекратить считывание параметров. В противном случае будут показаны ошибочные данные.

5.2.4. Установка единиц измерений температуры и скорости ветра

Последний сценарий касается установки единиц измерений.

1. Пользователь нажимает клавишу *Режим*.
2. Система выводит на экран сообщение *РЕЖИМ*.
3. Пользователь нажимает одну из клавиш *Ветер* или *Температура*; нажатия всех остальных клавиш (кроме клавиши *Пуск*) игнорируются.
4. Название выбранного параметра начинает мигать на экране.
5. Пользователь нажимает одну из клавиш /\ (*стрелка вверх*) или \/ (*стрелка вниз*), изменяя при этом единицу измерения параметра.
6. Система изменяет единицу измерения выбранного параметра.
7. Переход управления к пункту 3 или 5.

Замечание. Для прекращения работы в данном режиме пользователь нажимает клавишу *Пуск*. При этом экран дисплея возвращается в первоначальное состояние, и происходит переустановка единиц измерений параметров.

5.2.5. Включение системы

Последний основной сценарий относится к включению системы. При этом требуется обеспечить создание всех ее объектов в нужной последовательности и приведение их в стабильное начальное состояние.

1. Включение питания.
2. Создание датчиков; датчики с историей очищают «исторические» данные; датчики с трендом инициализируют алгоритм вычисления тренда.
3. Инициализация буфера клавиатуры, очистка его от случайной информации, вызванной помехами при включении питания.
4. Прорисовка статических элементов экрана.
5. Инициализация процесса опроса датчиков.

Постусловия:

- последние минимаксные значения параметров устанавливаются равными их первому замеру;
- время достижения минимакса считается равным времени первого замера;
- тренды температуры и давления равны нулю;
- главный автомат переходит в состояние работы.

Отметим, что задание постусловий определяет ожидаемое состояние системы после завершения сценария. Выполнение этого сценария обеспечивается совместной работой целой группы объектов, каждый из которых самостоятельно приводит себя в стабильное начальное состояние.

5.4. Автоматы

В этом разделе заключается основная часть проектирования системы — четкое описание ее поведения для последующей изоморфной реализации.

5.4.1. Автомат A0

Этот автомат реализует в себе сценарий 5, а также начала остальных четырех сценариев. На рис. 7, 8 изображены диаграмма связей автомата A0 и его граф переходов.

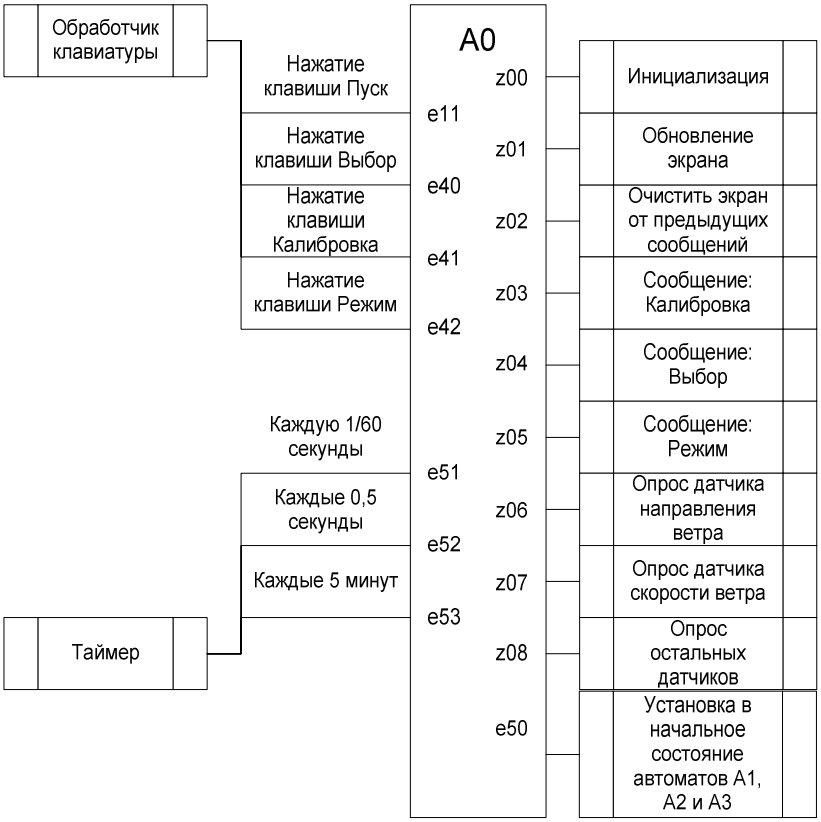

Рис. 7. Диаграмма связей автомата A0

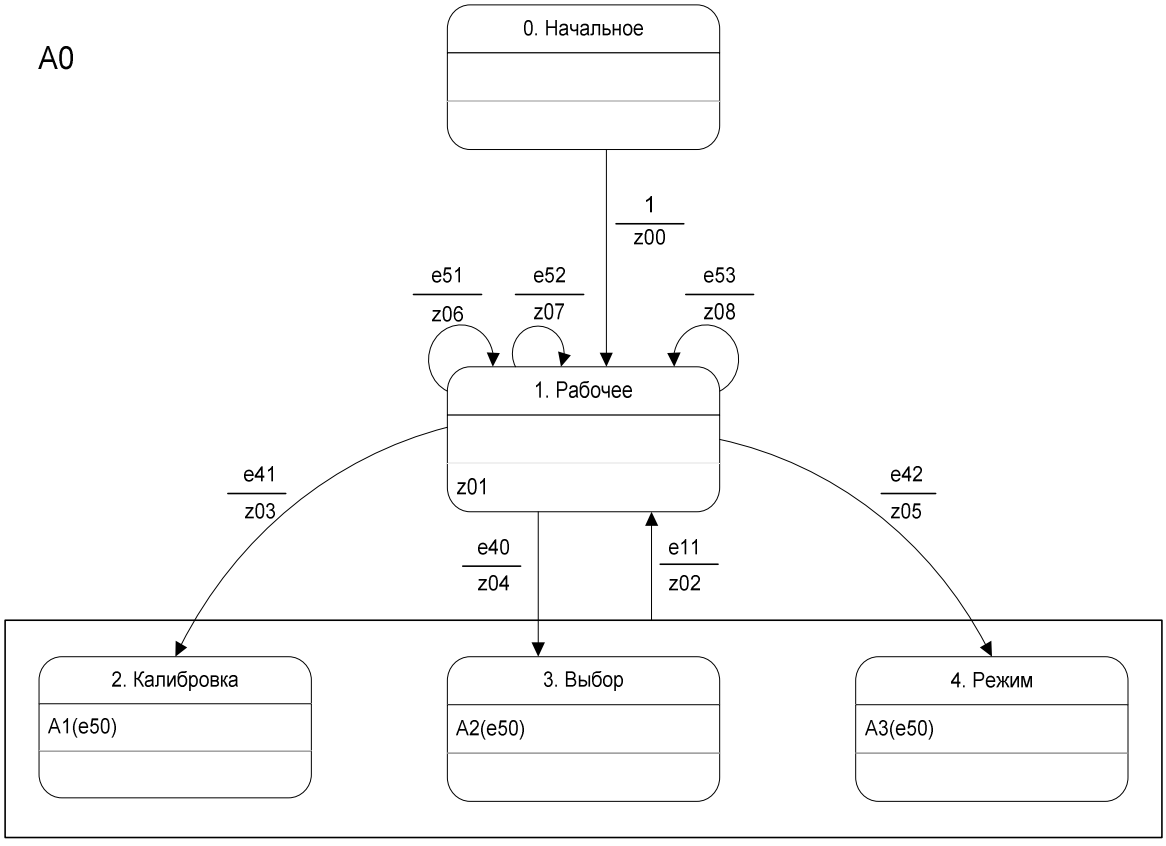

Рис. 8. Граф переходов автомата A0

5.4.2. Автомат A1

Автомат A1 соответствует сценарию 3. Он ответственен за калибровку датчиков. На рис. 9, 10 изображены диаграмма связей автомата A1 и его граф переходов.

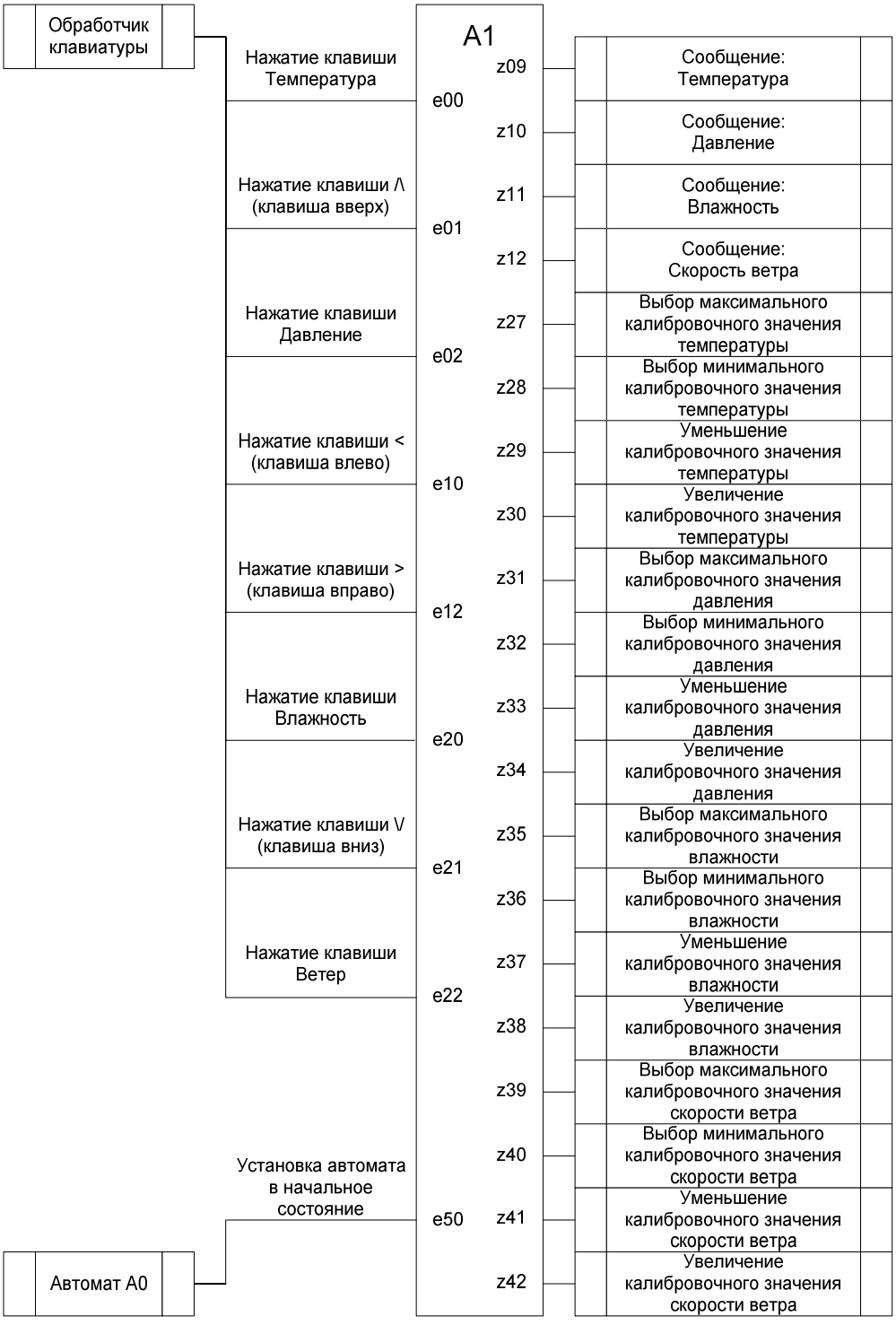

Рис. 9. Диаграмма связей автомата A1

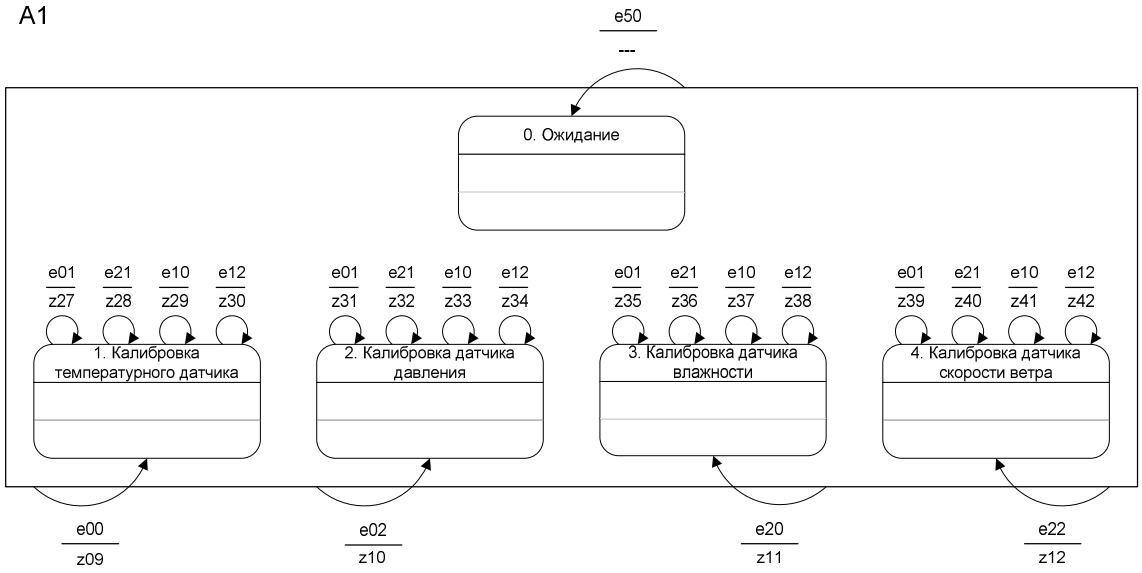

Рис. 10. Граф переходов автомата A1

5.4.3. Автомат A2

Автомат A2 соответствует сценариям 1 и 2. Он отвечает за все, что связано с нажатием клавиши «Выбор» — за показ минимальных и максимальных значений параметров, а также за установку времени и даты. На рис. 11 и рис. 12 изображены диаграмма связей автомата A2 и его граф переходов.

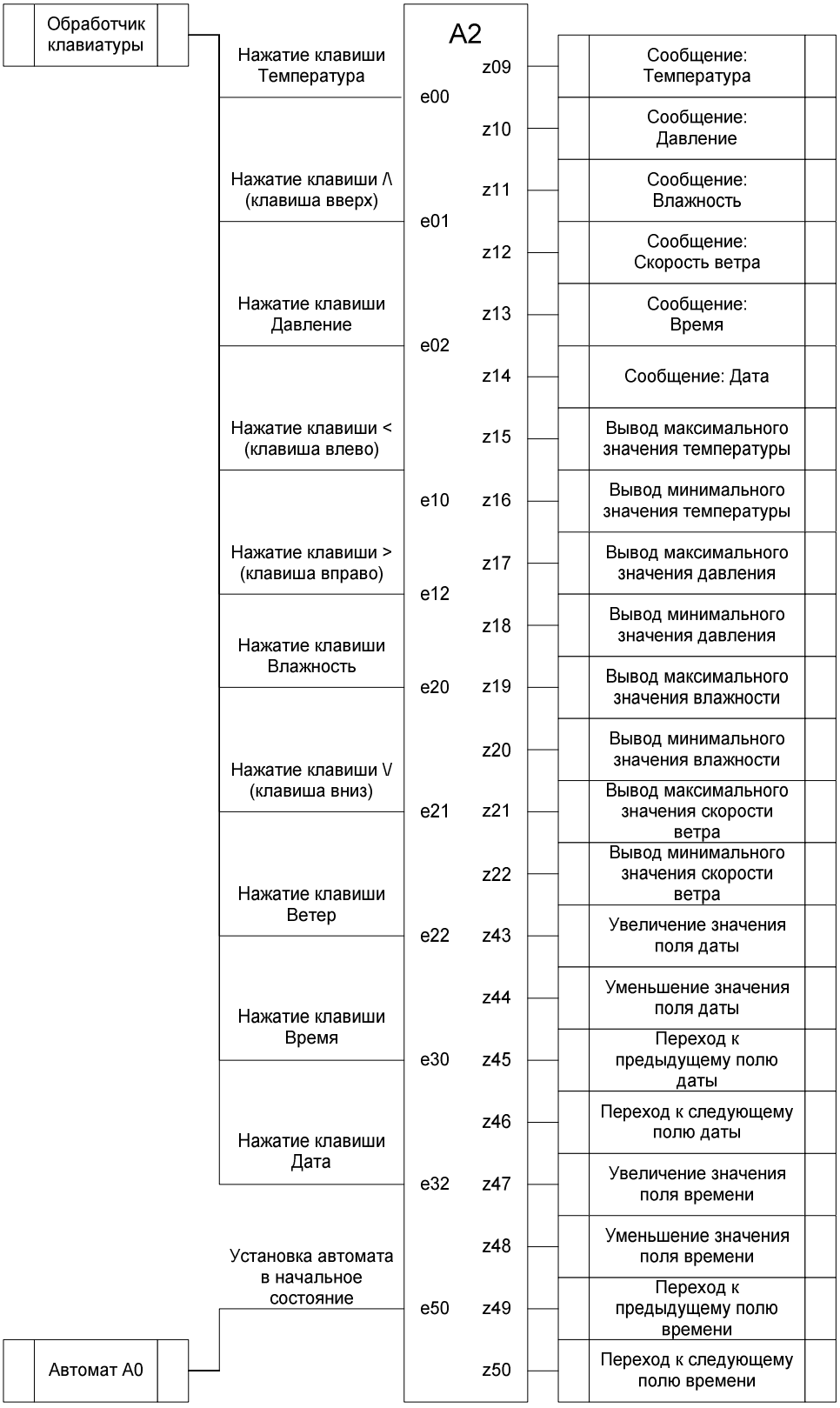

Рис. 11. Диаграмма связей автомата A2

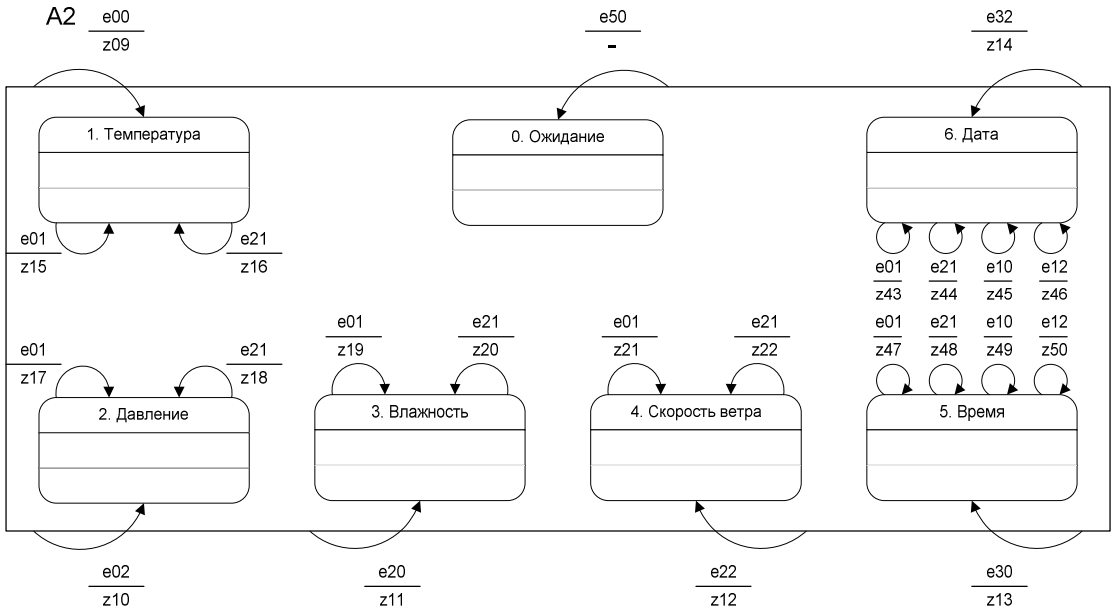

Рис. 12. Граф переходов автомата A2

5.4.4. Автомат A3

Автомат A3 соответствует оставшемуся сценарию 4 и отвечает за смену единиц измерения для температуры и скорости ветра. На рис. 13, 14 изображены диаграмма связей автомата A3 и его граф переходов.

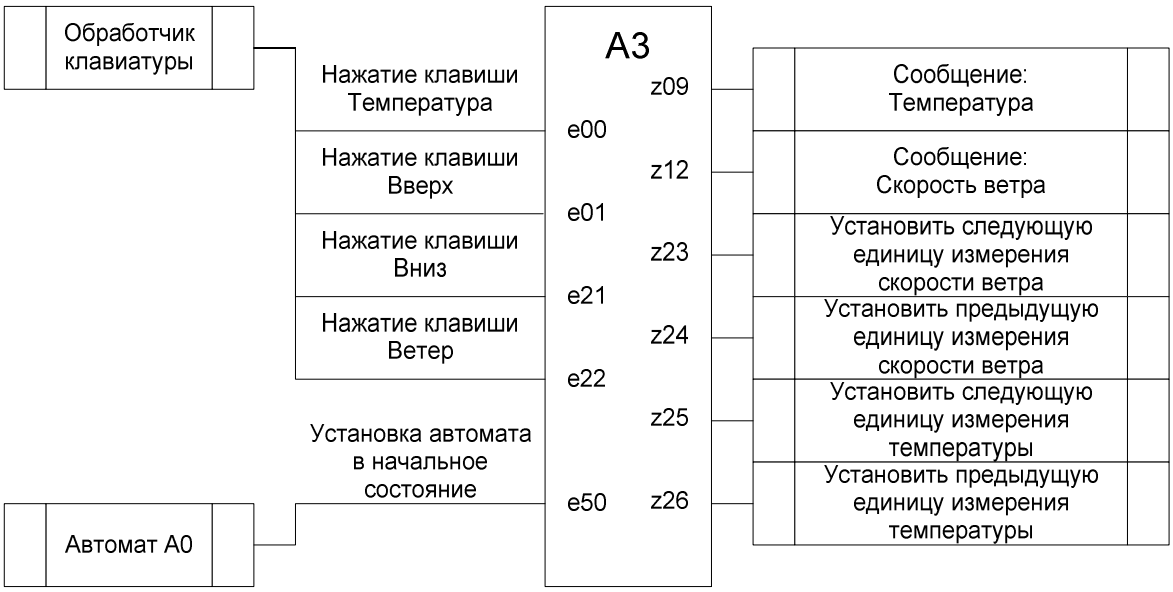

Рис. 13. Диаграмма связей автомата A3

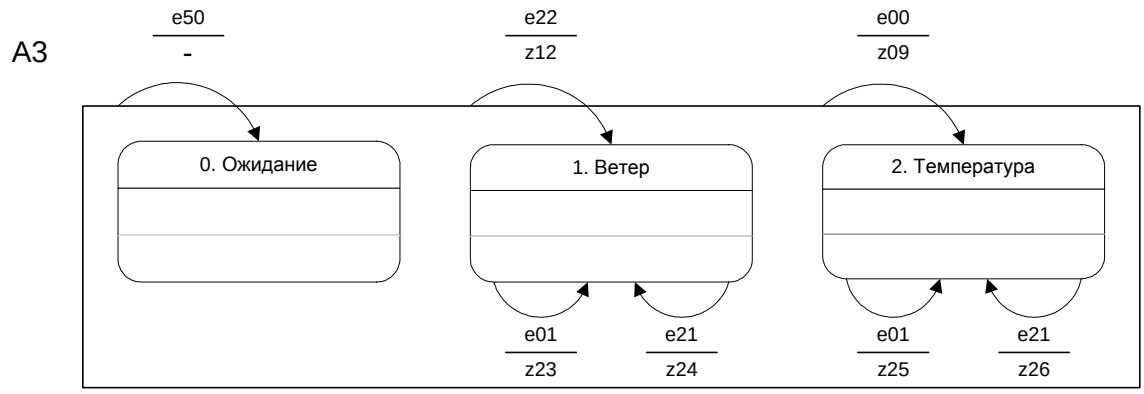

Рис. 14. Граф переходов автомата A3

6. Интерфейсы классов

> Никакая часть сложной системы не должна зависеть от внутреннего устройства какой-либо другой части.
>
> Д. Ингалс

> Инкапсуляция не спасает от глупости. Она, как отметил Страуструп, «защищает от ошибок, но не от жульничества».
>
> Г. Буч

В проекте используется идея технологии «клиент-сервер». Она применялась, например, в работе [2]. Между прочим, и Г. Буч в работе [1] не обходит стороной подобные аспекты реализации.

Клиентом называется любой объект, использующий ресурсы другого объекта (называемого *сервером*). Будем характеризовать поведение объекта услугами, которые он оказывает другим объектам, и операциями, которые он выполняет над другими объектами. Такой подход концентрирует внимание на внешних проявлениях объекта и приводит к идее, которую Б. Мейер назвал *контрактной моделью* программирования: внешнее проявление объекта рассматривается с точки зрения его контракта с другими объектами, в соответствии с этим должно быть выполнено и его внутреннее устройство (часто во взаимодействии с другими объектами). Контракт фиксирует все обязательства, которые объект-сервер имеет перед объектом-клиентом. Другими словами, этот контракт определяет *ответственность* объекта — то поведение, за которое он отвечает.

Каждая операция, предусмотренная этим контрактом, однозначно определяется ее формальными параметрами и типом возвращаемого значения. Полный набор операций, которые клиент может осуществлять над другим объектом, вместе с правильным порядком, в котором эти операции вызываются, называется *протоколом.* Протокол отражает все возможные способы, которыми объект может действовать или подвергаться воздействию. Он полностью определяет внешнее поведение абстракции со статической и динамической точки зрения.

Б. Лисков (женщина — профессор МТИ) прямо утверждает, что «абстракция будет работать только вместе с инкапсуляцией». Практически это означает наличие двух частей в классе: интерфейса и реализации. *Интерфейс* отражает внешнее поведение объекта, описывая абстракцию поведения всех объектов данного класса. Внутренняя *реализация*

описывает представление этой абстракции и механизмы достижения желаемого поведения объекта. Принцип разделения интерфейса и реализации соответствует сути вещей: в интерфейсной части собрано все, что касается взаимодействия данного объекта с любыми другими объектами. Реализация скрывает от других объектов все детали, не имеющие отношения к процессу взаимодействия объектов. К. Бритон и Д. Парнас назвали такие детали «тайнами абстракции»

Инкапсуляция — это процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение. Инкапсуляция используется для того, чтобы изолировать контрактные обязательства абстракции от их реализации.

В данном проекте вопрос интерфейсов особо остро стоит в отношении датчиков. Их иерархия изображена на рис. 15.

Рис. 15. Иерархия интерфейсов классов датчиков

7. Реализация

> Цель эволюции — наращивать и изменять реализацию, последовательно совершенствуя ее, чтобы в конечном счете создать готовую систему.
>
> Г. Буч

Рассмотрев несколько сценариев работы системы и убедившись в их правильности, можно начинать планирование процесса разработки. Разобьем работу на ряд этапов, результат каждого из которых будет являться основой для последующего:

- разработка программы, обладающей минимальными функциональными свойствами и осуществляющей мониторинг только одного датчика;
- создание иерархии датчиков;
- создание классов, ответственных за управление изображением на экране;
- создание классов, ответственных за работу пользовательского интерфейса.

В принципе, можно было бы изменить порядок этапов, но была выбрана именно такая последовательность, исходя из того, что наиболее сложная и рискованная часть работы должна выполняться в первую очередь.

Разработка минимальной версии программы заставляет, в первую очередь, смоделировать архитектуру «по вертикали», реализовав в усеченном варианте практически все ключевые абстракции. Эта задача несет в себе основной риск, ведь в процессе ее решения фактически проверяется правильность выбора ключевых абстракций, их роль и функции. Успешное создание раннего прототипа имеет важное в построении системы. В частности, сразу могут быть выявлены несоответствия между аппаратной и программной частями. Кроме того, будущие пользователи получат возможность уже на ранних этапах проекта оценить внешний вид и работу системы.

Итоговая диаграмма классов изображена на рис. 16.

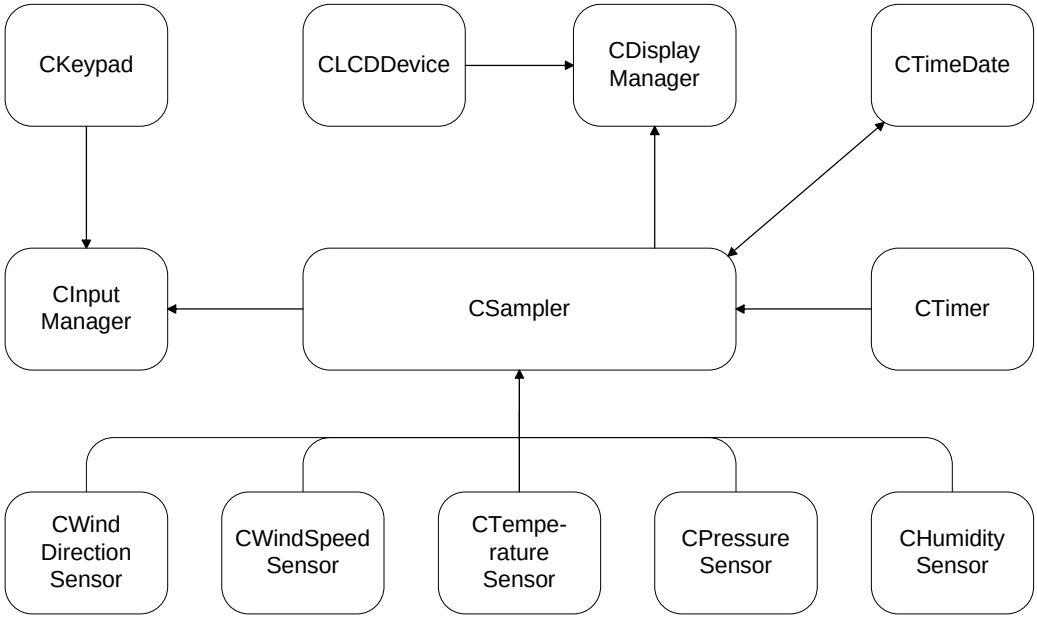

Рис. 16. Диаграмма классов

Можно заметить, что разбиение на классы полностью соответствует архитектуре, приведенной на рис. 2.

Отметим, что в проекте достаточно широко использовалась типизация, например, для того, чтобы не перепутать типы, соответствующие температуре и давлению. В общем случае, типизация — это способ защититься от использования объектов одного класса вместо другого или, по крайней мере, управлять таким использованием [1].

8. Заключение

Итак, кроме того, что была выполнена поставленная цель — реализован проект, было наглядно доказано, что процесс проектирования — это не только и не столько рисование абстрактных картинок, намекающих на будущую организацию системы, сколько трудоемкий процесс продумывания и досконального анализа взаимодействия компонент системы и работы системы в целом.

Важную роль в удачном проектировании сыграло применение автоматного программирования для точного описания поведения системы. Доказана неоценимая польза объектно-ориентированного проектирования при декомпозиции системы. Сочетание вышеуказанных технологий было достигнуто за счет замены диаграммы *statechart* на графы переходов соответствующих четырем автоматам.

Литература

1. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++. М.: Бином, СПб.: Невский диалект, 2000.
2. *Алексеев В.А., Ларионов А.В.* Сравнение программ управления кофеваркой «Mark 4 Special Coffee Maker», реализованных на основе нотации Буча и SWITCH-технологии. http://is.ifmo.ru/projects/coffee/.
3. *Страуструп Б.* Язык программирования C++. СПб.: Невский диалект, М.: БИНОМ, 1999.
4. *Кнут Д.* Искусство программирования. Т. 1. Основные алгоритмы. М.: Вильямс, 2000.
5. *Штучкин А.А., Шалыто А.А.* Совместное использование теории построения компиляторов и SWITCH-технологии (на примере построения калькулятора). http://is.ifmo.ru/projects/calc/.
6. *Наумов А.С., Шалыто А.А.* Система управления лифтом. http://is.ifmo.ru/projects/elevator/.
7. *Шалыто А.А., Туккель Н.И.* От тьюрингова программирования к автоматному // Мир ПК, 2002, № 2. http://is.ifmo.ru/works/turing/.
8. *Ахо А., Хопкрофт Д., Ульман Д.* Структуры данных и алгоритмы. М.: Вильямс, 2000.
9. *Мартин Р.* Быстрая разработка программ: принципы, примеры, практика. М.: Вильямс, 2004.

Приложение 1. Пример протокола работы автоматов

16:33:20{ A0: в состоянии 0 запущен с событием e42
16:33:20* z00. Инициализация
16:33:20T A0: перешел из состояния 0 в состояние 1
16:33:20} A0: завершил обработку события e42 в состоянии 1
16:33:20{ A0: в состоянии 1 запущен с событием e52
16:33:20* z07. Опрос датчика скорости ветра
16:33:20} A0: завершил обработку события e52 в состоянии 1
16:33:20{ A0: в состоянии 1 запущен с событием e51
16:33:20* z06. Опрос датчика направления ветра
16:33:20} A0: завершил обработку события e51 в состоянии 1
16:33:20{ A0: в состоянии 1 запущен с событием e53
16:33:20* z08. Опрос остальных датчиков
16:33:20} A0: завершил обработку события e53 в состоянии 1
16:33:21{ A0: в состоянии 1 запущен с событием e52
16:33:21* z07. Опрос датчика скорости ветра
16:33:21} A0: завершил обработку события e52 в состоянии 1
16:33:22{ A0: в состоянии 1 запущен с событием e52
16:33:22* z07. Опрос датчика скорости ветра
16:33:22} A0: завершил обработку события e52 в состоянии 1
16:33:22{ A0: в состоянии 1 запущен с событием e51
16:33:22* z06. Опрос датчика направления ветра
16:33:22} A0: завершил обработку события e51 в состоянии 1
16:33:23{ A0: в состоянии 1 запущен с событием e52
16:33:23* z07. Опрос датчика скорости ветра
16:33:23} A0: завершил обработку события e52 в состоянии 1
16:33:23{ A0: в состоянии 1 запущен с событием e42
16:33:23* z05. Сообщение: Режим
16:33:23{ A1: в состоянии 0 запущен с событием e50
16:33:23T A0: перешел из состояния 1 в состояние 4
16:33:23} A0: завершил обработку события e42 в состоянии 4
16:33:24{ A0: в состоянии 4 запущен с событием e02
16:33:24{ A3: в состоянии 0 запущен с событием e02
16:33:24} A3: завершил обработку события e02 в состоянии 0
16:33:24} A0: завершил обработку события e02 в состоянии 4
16:33:28{ A0: в состоянии 4 запущен с событием e00
16:33:28{ A3: в состоянии 0 запущен с событием e00
16:33:28* z09. Сообщение: Температура
16:33:28} A0: завершил обработку события e00 в состоянии 4
16:33:31{ A0: в состоянии 4 запущен с событием e01
16:33:31{ A3: в состоянии 2 запущен с событием e01
16:33:31* z25. Установить следующую единицу измерения температуры
16:33:31} A3: завершил обработку события e01 в состоянии 2
16:33:31} A0: завершил обработку события e01 в состоянии 4
16:33:33{ A0: в состоянии 4 запущен с событием e11
16:33:33{ A3: в состоянии 2 запущен с событием e11
16:33:33} A3: завершил обработку события e11 в состоянии 2
16:33:33* z02. Очистить экран от предыдущих сообщений
16:33:33T A0: перешел из состояния 4 в состояние 1
16:33:33} A0: завершил обработку события e11 в состоянии 1
16:33:33{ A0: в состоянии 1 запущен с событием e52
16:33:33* z07. Опрос датчика скорости ветра
16:33:33} A0: завершил обработку события e52 в состоянии 1
16:33:34{ A0: в состоянии 1 запущен с событием e52
16:33:34* z07. Опрос датчика скорости ветра
16:33:34} A0: завершил обработку события e52 в состоянии 1
16:33:34{ A0: в состоянии 1 запущен с событием e51
16:33:34* z06. Опрос датчика направления ветра
16:33:34} A0: завершил обработку события e51 в состоянии 1
16:33:35{ A0: в состоянии 1 запущен с событием e52
16:33:35* z07. Опрос датчика скорости ветра
16:33:35} A0: завершил обработку события e52 в состоянии 1
16:33:36{ A0: в состоянии 1 запущен с событием e52
16:33:36* z07. Опрос датчика скорости ветра
16:33:36} A0: завершил обработку события e52 в состоянии 1
16:33:36{ A0: в состоянии 1 запущен с событием e51
16:33:36* z06. Опрос датчика направления ветра
16:33:36} A0: завершил обработку события e51 в состоянии 1
16:33:36{ A0: в состоянии 1 запущен с событием e53
16:33:36* z08. Опрос остальных датчиков
16:33:36} A0: завершил обработку события e53 в состоянии 1
16:33:37{ A0: в состоянии 1 запущен с событием e52
16:33:37* z07. Опрос датчика скорости ветра
16:33:37} A0: завершил обработку события e52 в состоянии 1
16:33:38{ A0: в состоянии 1 запущен с событием e52
16:33:38* z07. Опрос датчика скорости ветра
16:33:38} A0: завершил обработку события e52 в состоянии 1
16:33:38{ A0: в состоянии 1 запущен с событием e51
16:33:38* z06. Опрос датчика направления ветра
16:33:38} A0: завершил обработку события e51 в состоянии 1
16:33:38{ A0: в состоянии 1 запущен с событием e40
16:33:38* z04. Сообщение: Выбор

16:33:38{ A1: в состоянии 2 запущен с событием e50
16:33:38T A0: перешел из состояния 1 в состояние 3
16:33:38} A0: завершил обработку события e40 в состоянии 3
16:33:39{ A0: в состоянии 3 запущен с событием e00
16:33:39{ A2: в состоянии 0 запущен с событием e00
16:33:39* z09. Сообщение: Температура
16:33:39} A0: завершил обработку события e00 в состоянии 3
16:33:40{ A0: в состоянии 3 запущен с событием e01
16:33:40{ A2: в состоянии 1 запущен с событием e01
16:33:40* z15. Вывод максимального значения температуры
16:33:40} A2: завершил обработку события e01 в состоянии 1
16:33:40} A0: завершил обработку события e01 в состоянии 3
16:33:42{ A0: в состоянии 3 запущен с событием e21
16:33:42{ A2: в состоянии 1 запущен с событием e21
16:33:42* z16. Вывод минимального значения температуры
16:33:42} A2: завершил обработку события e21 в состоянии 1
16:33:42} A0: завершил обработку события e21 в состоянии 3
16:33:43{ A0: в состоянии 3 запущен с событием e01
16:33:43{ A2: в состоянии 1 запущен с событием e01
16:33:43* z15. Вывод максимального значения температуры
16:33:43} A2: завершил обработку события e01 в состоянии 1
16:33:43} A0: завершил обработку события e01 в состоянии 3
16:33:45{ A0: в состоянии 3 запущен с событием e12
16:33:45{ A2: в состоянии 1 запущен с событием e12
16:33:45} A2: завершил обработку события e12 в состоянии 1
16:33:45} A0: завершил обработку события e12 в состоянии 3
16:33:46{ A0: в состоянии 3 запущен с событием e10
16:33:46{ A2: в состоянии 1 запущен с событием e10
16:33:46} A2: завершил обработку события e10 в состоянии 1
16:33:46} A0: завершил обработку события e10 в состоянии 3
16:33:46{ A0: в состоянии 3 запущен с событием e21
16:33:46{ A2: в состоянии 1 запущен с событием e21
16:33:46* z16. Вывод минимального значения температуры
16:33:46} A2: завершил обработку события e21 в состоянии 1
16:33:46} A0: завершил обработку события e21 в состоянии 3
16:33:48{ A0: в состоянии 3 запущен с событием e11
16:33:48{ A2: в состоянии 1 запущен с событием e11
16:33:48} A2: завершил обработку события e11 в состоянии 1
16:33:48* z02. Очистить экран от предыдущих сообщений
16:33:48T A0: перешел из состояния 3 в состояние 1
16:33:48} A0: завершил обработку события e11 в состоянии 1
16:33:48{ A0: в состоянии 1 запущен с событием e52
16:33:48* z07. Опрос датчика скорости ветра
16:33:48} A0: завершил обработку события e52 в состоянии 1
16:33:48{ A0: в состоянии 1 запущен с событием e51
16:33:48* z06. Опрос датчика направления ветра
16:33:48} A0: завершил обработку события e51 в состоянии 1
16:33:48{ A0: в состоянии 1 запущен с событием e53
16:33:48* z08. Опрос остальных датчиков
16:33:48} A0: завершил обработку события e53 в состоянии 1
16:33:49{ A0: в состоянии 1 запущен с событием e52
16:33:49* z07. Опрос датчика скорости ветра
16:33:49} A0: завершил обработку события e52 в состоянии

Приложение 2. Исходные коды

```
// Interfaces.h: интерфейсы всех классов.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_INTERFACES_H__192031B2_DDE5_40BC_B30F_48F7ACFA19DB__INCLUDED_)
#define AFX_INTERFACES_H__192031B2_DDE5_40BC_B30F_48F7ACFA19DB__INCLUDED_

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

class DisplayManager;
class InputManager;

#define INVISIBLE -54673426384.0f
#define EPS 1e-6

// единицы измерения температуры
enum TemperatureScale {
        TS_CELSIUS              = 0,
        TS_FAHRENHEIT   = 1
};

// единицы измерения скорости ветра
enum SpeedScale {
        SS_KMH  = 0,
        SS_MS   = 1
};

// Размер текста
enum TextSize {
        TS_NORMAL       = 0,
        TS_SMALL        = 1
};

// Размер линий
enum PenSize {
        PS_NORMAL       = 0,
        PS_BOLD         = 1
} ;

// Мерцание на экране определенных значений
enum Flashing {
        F_OFF                   = 0,
        F_MODE                  = 1,
        F_DATE                  = 2,
        F_TIME                  = 3,
        F_TEMPERATURE   = 4,
        F_PRESSURE              = 5,
        F_HUMIDITY              = 6,
        F_SPEED                 = 7
};

// Виртуальная клавиатура
enum Key {
   K_TEMP=00,    K_UP=01,      K_PRESSURE=02,
```

```
    K_LEFT=10,     K_RUN=11,       K_RIGHT=12,
    K_HUMIDITY=20, K_DOWN=21,      K_WIND=22,
    K_TIME=30,     K_EMPTY=31,     K_DATE=32,
    K_SELECT=40,   K_CALIBRATE=41, K_MODE=42
};

// Названия сенсоров
enum SensorName {
    S_WINDSPEED       = 0,
    S_WINDDIRECTION = 1,
    S_TEMPERATURE   = 2,
    S_HUMIDITY      = 3,
    S_PRESSURE      = 4
};



// Поддержка информации о текущем времени и о текущей дате
class TimeDate
{
public:
        TimeDate();
        virtual ~TimeDate();

        const char *currentTime();          // текущее время
        const char *currentDate();          // текущая дата
        void setFormat() {};                // устонавка формата
        void setHour(WORD hour);            // установка часов
        void setMinute(WORD min);           // установка минут
        void setSecond(WORD sec);           // установка секунд
        void setMonth(WORD month);          // установка месяца
        void setDay(WORD day);              // установка дня
        void setYear(WORD year);            // установка года

public:
        _SYSTEMTIME m_time;
};

// Поддержка информации о коде последней клавиши, нажатой на клавиатуре
class Keypad
{
public:
        Keypad();
        virtual ~Keypad();

        void keyPress(Key);                 // запоминаем клавишу, которая была нажата
        DWORD isInputPending() const;       // была ли нажата клавиша
    Key lastKeyPressed();                   // последняя нажатая клавиша

protected:
        Key  m_keyBuffer[100];              // буфер нажатых клавиш
        DWORD m_header;
};

// Класс датчиков
class Sensor
{
public:
        Sensor(SensorName, unsigned int id = 0);
```

```
        virtual ~Sensor();

    virtual float currentValue() = 0;
    virtual float rawValue() = 0;
    SensorName name() const;                    // Имя датчика
    unsigned int id() const;                    // ID датчика

protected:
        SensorName m_name;
        unsigned int m_id;
        float m_value;
};

// Управление выводом на экран графических элементов
class LCDDevice
{
public:
        LCDDevice       (HDC hdc);
        virtual ~LCDDevice();

        void drawText(int, int, const char *);              // рисовать текст
        void drawLine(int, int, int, int);                  // рисовать линию
        void drawCircle(int, int, int, int);                // рисовать окружность
        void setTextSize(TextSize textSize = TS_NORMAL);    // установка размера текста
        void setTextStyle(){};                              // установка начертания текста
        void setPenSize(PenSize penSize = PS_NORMAL);       // установка широны линии

        void drawRect(int, int, int, int);                  // рисовать прямоугольник

protected:
        HDC                 m_hdc;
        TextSize            m_textSize;
        PenSize             m_penSize;
        HPEN                m_hPen, m_hPenBold;
        HBRUSH              m_hBrush;
        HFONT               m_hFont, m_hFontSmall;
};

// Обеспечение линейной интерполяции значений, лежащих в известном интервале
class CalibratingSensor : public Sensor
{
public:
        CalibratingSensor(SensorName, unsigned int id = 0);
        virtual ~CalibratingSensor();

    void setLowValue(float);        // установка минимального значения
    void setHighValue(float);       // установка максимального значения
        float getLowValue();        // получение минимального значения
        float getHighValue()        // получение максимального значения
    virtual float currentValue();   // текущее значение
    virtual float rawValue() = 0;

protected:
        float m_calLowValue;
        float m_calHighValue;
};

//Поддержание информации о макимальных и минимальных значениях датчиков
class HistoricalSensor : public CalibratingSensor
```

```
{
public:
        HistoricalSensor(SensorName, unsigned int id = 0);
        virtual ~HistoricalSensor();

   float highValue() const;                    // Возвращает максимальное значение
   float lowValue() const;                     // Возвращает минимальное значение
   const char* timeOfHighValue() const;        // Возвращает время замера максимального значения
   const char* timeOfLowValue() const;         // Возвращает время замера минимального значения

   virtual float currentValue();               // Запоминание макимального и минимального значения датчика

protected:
        float m_lowValue;
        float m_highValue;
        char  m_lowTime[20];
        char  m_highTime[20];

        TimeDate m_timeDate;
};

// Определение тренда давления или температуры как наклона графика
// (в линейном приближении) изменения их значений за данный интервал времени
class TrendSensor : public HistoricalSensor
{
public:
        TrendSensor(SensorName, unsigned int id = 0);
        virtual ~TrendSensor();

   virtual float currentValue();       // текущее значение
   float trend() const;                // тренд

protected:
        void calculateTrend();
        float m_trend;
        float m_previousValue;
};

// Поддержание информации о текущей температуры
class TemperatureSensor : public TrendSensor
{
public:
        TemperatureSensor(unsigned int id = 0);
        virtual ~TemperatureSensor();

   float rawValue();                   // "сырое" значение температуры
   float currentTemperature();         // текущая температура
};

// Поддержание информации о текущем баромерическом давлении
class PressureSensor : public TrendSensor
{
public:
        PressureSensor(unsigned int = 0);
        virtual ~PressureSensor();

   float rawValue();                   // "сырое" значение давления
   float currentPressure();            // текущее давление
};
```

```
// Поддержание информации о текущей влажности,
// выраженной в процентах от 0% до 100%
class HumiditySensor : public HistoricalSensor
{
public:
        HumiditySensor(unsigned int id = 0);
        virtual ~HumiditySensor();

   float rawValue();          // "сырое" значение влажности
   float currentHumidity();  // текущая влажность
};

// Поддержание информации о текущей скорости ветра
class WindSpeedSensor : public HistoricalSensor
{
public:
        WindSpeedSensor(unsigned int id = 0);
        virtual ~WindSpeedSensor();

   float rawValue();                    // "сырое" значение скорости ветра
   float currentWindSpeed();            // текущая скорость ветра
};

// Поддержание информации о текущем направлении ветра,
// указываемом как точка на розе ветров
class WindDirectionSensor : public Sensor
{
public:
        WindDirectionSensor(unsigned int id = 0);
        virtual ~WindDirectionSensor();

        virtual float currentValue(); // текущее направление ветра
   float rawValue();                    // "сырое" значение направления ветра
   float currentWindDirection();        // текущее направление ветра
};

// Класс коллекции
template <class C> class Collection
{
public:
        Collection();
        virtual ~Collection();

        void add(int, C*);           // добавить объект в коллекцию
        C* get(int, int);            // полуить объект из коллекции
        int getNumber(int);          // количесиво объектов заданного типа

protected:
        C* m_pArray[100][100];
        int m_pCount[100];
        int m_nLen;
};

//Коллекция датчиков
class Sensors : protected Collection <Sensor>
{
public:
   Sensors() ;
```

```
    virtual ~Sensors();

    void addSensor(Sensor& SensorName);                 // Добавление датчика в коллекцию
    unsigned int numberOfSensors();                     // Возвращает количества датчиков в коллекции
    unsigned numberOfSensors(SensorName);               // Возвращает количество датчиков определенного типа
    Sensor* sensor(SensorName, unsigned int id = 0);    // Возвращает датчик из коллекции
};

// Обеспечение прерываний и диспетчеризация функций обратного вызова
class Timer
{
public:
    void setCallback( VOID (CALLBACK*)(HWND, UINT, UINT, DWORD) );     // установка функций обратоного вызова
    void startTiming( void );                                          // запуск таймера
    unsigned int getNumberOfTicks();                                   // возвращает количество "тиков" таймера

protected:
        VOID (CALLBACK*m_func)(HWND, UINT, UINT, DWORD);
        unsigned int m_tickNumber;
};

//Класс-агент: контроль за процессом считывания параметров
class Sampler
{
public:
    Sampler(Sensors&, DisplayManager&, InputManager&);
    ~Sampler();

        unsigned int samplinRate(SensorName);               // Возвращает период опроса датчика
        void setSamplinRate(SensorName, unsigned int);      // Задание периода опроса датчиков
    void sample (unsigned int);                 // Проверяет, пришло ли время считывать информацию с датчика

protected:
    Sensors&        m_sensors;
    DisplayManager& m_displayManager;
    InputManager& m_inputManager;
        unsigned int    m_samplinRate[7];
        TimeDate            m_timeDate;
};

// Диспетчиризайия  команд пользователя
class InputManager
{
public:
        InputManager(Keypad&);
        virtual ~InputManager();

        int isRunning();            // проверяет является ли текущее состояние RUNNING
        void processKeyPress();     // обработка сигналов с клавиатуры

protected:
        Keypad&                 m_keypad;
        TimeDate                m_timeDate;
};

// Организация отображения параметров на экране дисплея
class DisplayManager
{
public:
```

```
DisplayManager(HWND hwnd);
virtual ~DisplayManager();

void clear();                                          // очистка экрана
void refresh();                                        // обновление экрана
void display(Sensor &);                                // отображение сенсора
void drawStaticItems();                                // рисование статических элементов
void displayTime(const char *);                        // вывод времени
void displayDate(const char *);                        // вывод даты
void displayTemperature(float, unsigned int id = 0);   // вывод температуры
void displayHumidity(float, unsigned int id = 0);      // вывод влажности
void displayPressure(float, unsigned int id = 0);      // вывод давления
void displayWindChill(float, unsigned int id = 0);     // вывод коэффициента жесткости погоды
void displayDewPoint(float, unsigned int id = 0);      // вывод точки росы
void displayWindSpeed(float, unsigned int id = 0);     // вывод скорости ветра
void displayWindDirection(unsigned int, unsigned int id = 0);
                                                       // вывод направления ветра
void displayHighLow(float, const char *, SensorName, unsigned int id = 0) {};
                                                       // вывод максимальных и минимальных значений
void setTemperatureScale(TemperatureScale) {           // вывод единицы измерения температуры
        m_ts = (m_ts == TS_CELSIUS) ? TS_FAHRENHEIT : TS_CELSIUS;
};
void setSpeedScale(SpeedScale) {                       // вывод единицы измерения скорости ветра
        m_ss = (m_ss == SS_KMH) ? SS_MS : SS_KMH;
};
void flashLabel(Flashing flash) {                      // вывод мерцающей надписи
        m_flash =  flash;
};
void displayMode(const char *);                        // вывод заголовка дисплея

void displayValueDate(const char *date0, const char *date1) {

        // сохранение текущего значения даты
        strcpy(m_cDate[0], date0);
        strcpy(m_cDate[1], date1);
};
void displayValueTime(const char *time0, const char *time1) {

        // сохранение текущего значения времени
        strcpy(m_cTime[0], time0);
        strcpy(m_cTime[1], time1);
};
void displayValueTemperature(const float temp) {       // сохранение текущего значения температуры
        m_cTemperature = temp;
};
void displayValuePressure(const float press) {         // сохранение текущего значения давления
        m_cPressure = press;
};
void displayValueHumidity(const float hum) {           // сохранение текущего значения влажности
        m_cHumidity = hum;
};
void displayValueWind(const float speed) {             // сохранение текущего значения скорости ветра
        m_cSpeed = speed;
};
void refresh(HDC hdc);
void blink() {
        m_visible = !m_visible;
};
```

```
protected:
        void displaySensorsValue();

        HWND            m_hwnd;
        HDC                     m_hdc;
        LCDDevice *m_lcd;
        Flashing m_flash;
        bool            m_visible;
        float           m_temperature, m_pressure, m_humidity, m_speed, m_direction;
        char            m_date[20], m_time[20];
        char            m_mode[20];
        UINT_PTR        m_timerID;
        char            m_cTime[2][20], m_cDate[2][20];
        float           m_cTemperature, m_cPressure, m_cHumidity, m_cSpeed;
        float           m_windChill;
        float           m_dewPoint;
        TemperatureScale m_ts;
        SpeedScale      m_ss;


};
#endif // !defined(AFX_INTERFACES_H__192031B2_DDE5_40BC_B30F_48F7ACFA19DB__INCLUDED_)

// CalibratingSensor.cpp
// Обеспечение линейной интерполяции значений, лежащих в известном интервале
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"

CalibratingSensor::CalibratingSensor(SensorName sensorName, unsigned int id/* = 0*/) :
   Sensor(sensorName, id)
{
   m_calLowValue  = 0;
   m_calHighValue = 1;
};

CalibratingSensor::~CalibratingSensor() {
};

// установка минимального значения
void CalibratingSensor::setLowValue(float value) {
   m_calLowValue = value;
};

// установка максимального значения
void CalibratingSensor::setHighValue(float value) {
   m_calHighValue = value;
};

// текущее значение
float CalibratingSensor::currentValue() {
   return (m_calHighValue-m_calLowValue)*rawValue() + m_calLowValue;
};

// получение минимального значения
float CalibratingSensor::getLowValue() {
   return m_calLowValue;
};
```

```
// получение максимального значения
float CalibratingSensor::getHighValue() {
   return m_calHighValue;
};

// DisplayManager.cpp
// Организация отображения параметров на экране дисплея
//////////////////////////////////////////////////////////////////////////

#include "stdafx.h"

static DisplayManager *pcmdm;

// мигание надписи
VOID CALLBACK flash(HWND hwnd,  UINT uMsg, UINT idEvent,  DWORD dwTime) {
        pcmdm->blink();
        pcmdm->refresh();
}

DisplayManager::DisplayManager(HWND hwnd) :
        m_hwnd(hwnd)
{
        m_flash = F_OFF;
        m_visible = true;

        strcpy(m_date,"03-20-98");
        strcpy(m_time, "13:56:42");
        strcpy(m_mode, "Выбор");
        m_temperature = 14;
        m_pressure = 29.96f;
        m_humidity = 38;
        m_speed = 7;
        m_direction = 110;
        m_ts = TS_CELSIUS;
        m_ss = SS_KMH;

        pcmdm = this;
        m_timerID = SetTimer(NULL, NULL, 300, flash);
}

DisplayManager::~DisplayManager() {
        if (m_timerID != 0) KillTimer(NULL, m_timerID);
}

// очистка экрана
void DisplayManager::clear() {
        m_lcd->drawRect(-5, -5, 377, 170);
}

// обновление экрана
void DisplayManager::refresh() {
        RECT rt;
        GetClientRect(m_hwnd, &rt);
        InvalidateRect(m_hwnd, &rt, FALSE);
}

// отображение сенсора
void DisplayManager::display(Sensor &sensor) {
        switch (sensor.name())      {
```

```
		case S_WINDSPEED:
			displayWindSpeed(sensor.currentValue());
			break;
		case S_WINDDIRECTION:
			displayWindDirection((int)(sensor.currentValue()));
			break;
		case S_TEMPERATURE:
			displayTemperature(sensor.currentValue());
			{
				TemperatureSensor *temp = (TemperatureSensor*)&sensor;
				displayWindChill(temp->trend());
			}
			break;
		case S_HUMIDITY:
			displayHumidity(sensor.currentValue());
			break;
		case S_PRESSURE:
			displayPressure(sensor.currentValue());
			{
				PressureSensor *temp = (PressureSensor*)&sensor;
				displayDewPoint(temp->trend());
			}
			break;
		default:
			break;
		};
};

// рисование статических элементов
void DisplayManager::drawStaticItems () {
		m_lcd->setPenSize();
		m_lcd->drawCircle(240, 30, 340, 130);

		m_lcd->drawLine(290, 30, 290, 20);  m_lcd->drawText(285, 5, "С");
		m_lcd->drawLine(290, 130, 290, 140);m_lcd->drawText(285, 140, "Ю");
		m_lcd->drawLine(240, 80, 230, 80);  m_lcd->drawText(223, 72, "З");
		m_lcd->drawLine(340, 80, 350, 80);  m_lcd->drawText(350, 72, "В");

		m_lcd->setTextSize();
		if (m_visible || m_flash != F_DATE) m_lcd->drawText(75, 25,  "ДАТА");
		if (m_visible || m_flash != F_TIME) m_lcd->drawText(62, 50,  "ВРЕМЯ");
		if (m_visible || m_flash != F_TEMPERATURE) m_lcd->drawText(10, 75,  "ТЕМПЕРАТУРА");
		if (m_visible || m_flash != F_PRESSURE) m_lcd->drawText(33, 100, "ДАВЛЕНИЕ");
		if (m_visible || m_flash != F_HUMIDITY) m_lcd->drawText(20, 125, "ВЛАЖНОСТЬ");
		if (m_visible || m_flash != F_SPEED) m_lcd->drawText(252, 85, "СКОРОСТЬ");

		m_lcd->setTextSize(TS_SMALL);
		m_lcd->drawLine(290+35, 80-35, 290+40, 80-40);m_lcd->drawText(290+40, 80-50, "СВ");
		m_lcd->drawLine(290-36, 80-36, 290-40, 80-40);m_lcd->drawText(290-57, 80-50, "СЗ");
		m_lcd->drawLine(290-35, 80+35, 290-40, 80+40);m_lcd->drawText(290-57, 80+40, "ЮЗ");
		m_lcd->drawLine(290+35, 80+35, 290+40, 80+40);m_lcd->drawText(290+40, 80+40, "ЮВ");

}

// вывод времени
void DisplayManager::displayTime(const char *time) {
		strcpy(m_time, time);
}
```

```
// вывод даты
void DisplayManager::displayDate(const char *date) {
	strcpy(m_date, date);
}

// вывод температуры
void DisplayManager::displayTemperature(float value , unsigned int /*id = 0*/) {
	m_temperature = value;
}

// вывод влажности
void DisplayManager::displayHumidity(float value, unsigned int /*id = 0*/) {
	m_humidity = value;
}

// вывод давления
void DisplayManager::displayPressure(float value, unsigned int /*id = 0*/) {
	m_pressure = value;
}

// вывод коэффициента жесткости погоды
void DisplayManager::displayWindChill(float value, unsigned int /*id = 0*/) {
	m_windChill = value;
};

// вывод точки росы
void DisplayManager::displayDewPoint(float value, unsigned int /*id = 0*/) {
	m_dewPoint = value;
};

// вывод скорости ветра
void DisplayManager::displayWindSpeed(float dir, unsigned int /*id = 0*/) {
	m_speed = dir;
}

// вывод направления ветра
void DisplayManager::displayWindDirection(unsigned int dir, unsigned int /*id = 0*/) {
	m_direction = (float)(dir/**0.0174532925199432*/);
}

// вывод заголовка дисплея
void DisplayManager::displayMode(const char *mode) {
	strcpy(m_mode, mode);
}

void DisplayManager::refresh(HDC hdc) {

	m_hdc = hdc;
	m_lcd   = new LCDDevice(hdc);
	clear();
	drawStaticItems();
	if (m_flash == F_OFF) displaySensorsValue();
	else if (m_visible) {
		char str[20];

		m_lcd->setTextSize(TS_SMALL);
		m_lcd->drawText(125, 5, m_mode);
		m_lcd->setTextSize();
		switch (m_flash) {
```

```
                case F_DATE:
                        m_lcd->drawText(125, 25, m_cDate[0]);
                        break;
                case F_TIME:
                        m_lcd->drawText(125, 50, m_cTime[0]);
                        break;
                case F_TEMPERATURE:
                        if (fabs(m_cTemperature-INVISIBLE) < EPS) {
                                if (m_ts == TS_CELSIUS) m_lcd->drawText(125, 75, "°C");
                                else m_lcd->drawText(125, 75, "°F");
                                break;
                        }
                        if (m_ts == TS_CELSIUS) sprintf(str, "%.2f °C", m_cTemperature);
                        else sprintf(str, "%.2f °F", m_cTemperature);
                        m_lcd->drawText(125, 75, str);
                        break;
                case F_PRESSURE:
                        if (fabs(m_cPressure-INVISIBLE) < EPS) break;
                        sprintf(str, "%.0f мм.рт.ст.", m_cPressure);
                        m_lcd->drawText(125, 100, str);
                        break;
                case F_HUMIDITY:
                        if (fabs(m_cHumidity-INVISIBLE) < EPS) break;
                        sprintf(str, "%.0f %%", m_cHumidity);
                        m_lcd->drawText(125, 125, str);
                        break;
                case F_SPEED:
                        {
                                m_lcd->setTextSize(TS_SMALL);
                                if (m_ss == SS_KMH) sprintf(str, "%.1f км/ч", m_cSpeed);
                                else sprintf(str, "%.1f м/с", m_cSpeed);
                                UINT oldTextAlign = SetTextAlign(m_hdc, TA_CENTER);
                                if (fabs(m_cSpeed-INVISIBLE) > EPS) m_lcd->drawText(290, 65, str);
                                else {
                                        if (m_ss == SS_KMH) m_lcd->drawText(290, 65, "км/ч");
                                        else m_lcd->drawText(290, 65, "м/с");
                                }
                                SetTextAlign(m_hdc, oldTextAlign);
                                m_lcd->setTextSize();
                        }
                        break;
        }
        if ((m_flash != F_TIME)&&(m_flash != F_DATE))
        {
                m_lcd->drawText(125, 25, m_cDate[0]);
                m_lcd->drawText(125, 50, m_cTime[0]);
        }
}
else {
        m_lcd->setTextSize();
        switch (m_flash) {
                case F_DATE:
                        m_lcd->drawText(125, 25, m_cDate[1]);
                        break;
                case F_TIME:
                        m_lcd->drawText(125, 50, m_cTime[1]);
                        break;
                default:
                        m_lcd->drawText(125, 25, m_cDate[0]);
```

```
                                        m_lcd->drawText(125, 50, m_cTime[0]);
                                        break;
                                }

                        }
                        delete m_lcd;
}

void DisplayManager::displaySensorsValue() {
                        m_lcd->setTextSize();
                        m_lcd->drawText(125, 50, m_time);
                        m_lcd->drawText(125, 25, m_date);

                        char str[20];
                        if (m_ts == TS_CELSIUS) sprintf(str, "%.1f °C", m_temperature);
                        else sprintf(str, "%.1f °F", m_temperature);
                        m_lcd->drawText(125, 75, str);

                        sprintf(str, "%.0f %%", m_humidity);
                        m_lcd->drawText(125, 125, str);

                        sprintf(str, "%.0f мм.рт.ст.", m_pressure);
                        m_lcd->drawText(125, 100, str);

                        m_lcd->setTextSize(TS_SMALL);
                        if (m_ss == SS_KMH) sprintf(str, "%.1f км/ч", m_speed);
                        else sprintf(str, "%.1f м/с", m_speed);
                        UINT oldTextAlign = SetTextAlign(m_hdc, TA_CENTER);
                        m_lcd->drawText(290, 65, str);
                        SetTextAlign(m_hdc, oldTextAlign);

                        m_lcd->setPenSize(PS_BOLD);
                        double sina = sin(-m_direction);
                        double cosa = cos(-m_direction);
                        m_lcd->drawLine((int)(290+cosa*40), (int)(80+sina*40), (int)(290+cosa*60), (int)(80+sina*60));

                        m_lcd->setPenSize();
                        if (m_windChill > 0) {
                                m_lcd->drawRect(210, 88-(int)(m_windChill*10), 220, 89);
                        }
                        else {
                                m_lcd->drawRect(210, 88, 220, 88-(int)(m_windChill*10));
                        }
                        m_lcd->drawLine(205, 88, 225, 88);
                        if (m_dewPoint > 0) {
                                m_lcd->drawRect(210, 113-(int)(m_dewPoint*10), 220, 114);
                        }
                        else {
                                m_lcd->drawRect(210, 113, 220, 113-(int)(m_dewPoint*10));
                        }
                        m_lcd->drawLine(205, 113, 225, 113);
}

// HistoricalSensor.cpp
// Поддержание информации о макимальных и минимальных значениях датчиков
////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Interfaces.h"
```

```
HistoricalSensor::HistoricalSensor(SensorName sensorName, unsigned int id/* = 0*/) :
   CalibratingSensor(sensorName, id)
{
   m_lowValue  = 1000000;
   m_highValue = -1000000;
   strcat(m_lowTime,  "00:00:00 01-01-79");
   strcat(m_highTime, "00:00:00 01-01-79");
};

HistoricalSensor::~HistoricalSensor()
{
};

// Возвращает максимальное значение
float HistoricalSensor::highValue() const
{
   return m_highValue;
};

// Возвращает минимальное значение
float HistoricalSensor::lowValue() const
{
   return m_lowValue;
};

// Возвращает время замера максимального значения
const char* HistoricalSensor::timeOfHighValue() const
{
   return m_highTime;
};

// Возвращает время замера минимального значения
const char* HistoricalSensor::timeOfLowValue() const
{
   return m_lowTime;
};

// Запоминание макимального и минимального значения датчика
float HistoricalSensor::currentValue()
{
   float value = CalibratingSensor::currentValue();
   if (m_lowValue >= value) {
      m_lowValue = value;
      strcpy(m_lowTime, m_timeDate.currentTime());
      strcpy(&m_lowTime[9], m_timeDate.currentDate());
   }
   if (m_highValue <= value) {
      m_highValue = value;
      strcpy(m_highTime, m_timeDate.currentTime());
      strcpy(&m_highTime[9], m_timeDate.currentDate());
   }
   return value;
};

// HistoricalSensor.cpp
// Поддержание информации о макимальных и минимальных значениях датчиков
////////////////////////////////////////////////////////////////////////
```

```
#include "stdafx.h"
#include "Interfaces.h"

HistoricalSensor::HistoricalSensor(SensorName sensorName, unsigned int id/* = 0*/) :
   CalibratingSensor(sensorName, id)
{
   m_lowValue  = 1000000;
   m_highValue = -1000000;
   strcat(m_lowTime,  "00:00:00 01-01-79");
   strcat(m_highTime, "00:00:00 01-01-79");
};

HistoricalSensor::~HistoricalSensor()
{
};

// Возвращает максимальное значение
float HistoricalSensor::highValue() const
{
   return m_highValue;
};

// Возвращает минимальное значение
float HistoricalSensor::lowValue() const
{
   return m_lowValue;
};

// Возвращает время замера максимального значения
const char* HistoricalSensor::timeOfHighValue() const
{
   return m_highTime;
};

// Возвращает время замера минимального значения
const char* HistoricalSensor::timeOfLowValue() const
{
   return m_lowTime;
};

// Запоминание макимального и минимального значения датчика
float HistoricalSensor::currentValue()
{
   float value = CalibratingSensor::currentValue();
   if (m_lowValue >= value) {
      m_lowValue = value;
      strcpy(m_lowTime, m_timeDate.currentTime());
      strcpy(&m_lowTime[9], m_timeDate.currentDate());
   }
   if (m_highValue <= value) {
      m_highValue = value;
      strcpy(m_highTime, m_timeDate.currentTime());
      strcpy(&m_highTime[9], m_timeDate.currentDate());
   }
   return value;
};

// InputManager.cpp
// Диспетчиризайия  команд пользователя
```

```
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "auto.h"

InputManager::InputManager(Keypad& keypad) :
        m_keypad(keypad)
{
};

InputManager::~InputManager() {
};

// проверяет является ли текущее состояние RUNNING
int InputManager::isRunning() {
        return a_y0 == 1;
}

// обработка сигналов с клавиатуры
void InputManager::processKeyPress() {
        if (m_keypad.isInputPending()) {
                Key key = m_keypad.lastKeyPressed();
                A0(key);
   };
};

// Keypad.cpp
// Поддержка информации о коде последней клавиши, нажатой на клавиатуре
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"

Keypad::Keypad() :
   m_header(0) {
};

Keypad::~Keypad(){
};

// запоминаем клавишу, которая была нажата
void Keypad::keyPress(Key key) {
   if (m_header > 100) {
      return;
   }
   m_keyBuffer[m_header++] = key;
};

// была ли нажата клавиша
DWORD Keypad::isInputPending() const {
   return m_header;
};

// последняя нажатая клавиша
Key Keypad::lastKeyPressed() {
   if (m_header > 0) {
      return m_keyBuffer[--m_header];
   }
   return K_EMPTY;
};
```

```
// LCDDevice.cpp
// Управление выводом на экран графических элементов
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"

LCDDevice::LCDDevice(HDC hdc) :
   m_hdc(hdc),
   m_textSize(TS_NORMAL),
   m_penSize(PS_NORMAL)
{

   m_hPen      = CreatePen(PS_SOLID, 1, RGB(80, 160, 0));
   m_hPenBold  = CreatePen(PS_SOLID, 3, RGB(80, 160, 0));
   m_hBrush    = CreateSolidBrush(RGB(0, 0, 0));
   m_hFont     = CreateFont(20, 7, GM_COMPATIBLE, GM_COMPATIBLE, FW_BOLD,
      FALSE, FALSE, FALSE, RUSSIAN_CHARSET,
      OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH, NULL);
   m_hFontSmall= CreateFont(15, 5, GM_COMPATIBLE, GM_COMPATIBLE, FW_NORMAL,
      FALSE, FALSE, FALSE, RUSSIAN_CHARSET,
      OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH, NULL);

   SetTextColor(hdc, RGB(80, 160, 0));
   SetBkColor(hdc, RGB(0, 0, 0));
   SelectObject(hdc, m_hBrush);
};

LCDDevice::~LCDDevice() {
   DeleteObject(m_hPen);
   DeleteObject(m_hPenBold);
   DeleteObject(m_hBrush);
   DeleteObject(m_hFont);
   DeleteObject(m_hFontSmall);
};

// рисовать текст
void LCDDevice::drawText(int x, int y, const char *text) {
   if (m_textSize == TS_NORMAL) SelectObject(m_hdc, m_hFont);
   else SelectObject(m_hdc, m_hFontSmall);
   TextOut(m_hdc, x, y, text, strlen(text));
};

// рисовать линию
void LCDDevice::drawLine(int x1, int y1, int x2, int y2) {
          if (m_penSize == PS_NORMAL) {
                    SelectObject(m_hdc, m_hPen);
          }
          else {
                    SelectObject(m_hdc, m_hPenBold);
          }
   MoveToEx(m_hdc, x1, y1, NULL);
   LineTo(m_hdc, x2, y2);
};

// рисовать окружность
void LCDDevice::drawCircle(int x1, int y1, int x2, int y2) {
          if (m_penSize == PS_NORMAL) {
                    SelectObject(m_hdc, m_hPen);
```

```
		}
		else {
			SelectObject(m_hdc, m_hPenBold);
		}
	Ellipse(m_hdc, x1, y1, x2, y2);
};

// установка размера текста
void LCDDevice::setTextSize(TextSize size /*= TS_NORMAL*/) {
	m_textSize = size;
};

// установка широны линии
void LCDDevice::setPenSize(PenSize size /*= PS_NORMAL*/ ) {
	m_penSize = size;
};

// рисовать прямоугольник
void LCDDevice::drawRect(int x1, int y1, int x2, int y2) {
		if (m_penSize == PS_NORMAL) {
			SelectObject(m_hdc, m_hPen);
		}
		else {
			SelectObject(m_hdc, m_hPenBold);
		}
	Rectangle(m_hdc, x1, y1, x2, y2);
};

// PressureSensor.cpp
// Поддержание информации о текущем баромерическом давлении
////////////////////////////////////////////////////////////////////////

#include "stdafx.h"

PressureSensor::PressureSensor(unsigned int id/* = 0*/) :
	TrendSensor(S_PRESSURE, id)
{
};

PressureSensor::~PressureSensor() {
};

// "сырое" значение давления
float PressureSensor::rawValue() {
	m_value += (float)((1 - (rand() / 16384.0))/2);
		if (m_value < 0) m_value = 0;
		if (m_value > 1) m_value = 1;
		return m_value;//(float)(rand() / 32767.0);
};

// текущее давление
float PressureSensor::currentPressure() {
	return currentValue();
};

// Sampler.cpp
// Класс-агент: контроль за процессом считывания параметров
////////////////////////////////////////////////////////////////////////
```

```
#include "stdafx.h"
#include "Interfaces.h"
#include "auto.h"

Sampler::Sampler(Sensors& sensors, DisplayManager& displayManager, InputManager& inputManager):
   m_sensors(sensors),
   m_displayManager(displayManager),
   m_inputManager(inputManager)
{
}

Sampler::~Sampler()
{
}

// Возвращает период опроса датчика
unsigned int Sampler::samplinRate(SensorName name)
{
   return m_samplinRate[name];
};

// Задание периода опроса датчиков
void Sampler::setSamplinRate(SensorName name, unsigned int tick)
{
      m_samplinRate[name] = tick;
};
// Проверяет, пришло ли время считывать информацию с датчика
void Sampler::sample(unsigned int tick)
{
   static bool state = false;

         // обрабатываем значения с клавиатуры
   m_inputManager.processKeyPress();

         // Если вышли из состояния RUNNING, то прекращаем опрос датчиков
   if (m_inputManager.isRunning()) {
      if (!state) {
         state = true;
      }
   }
   else state = false;

   if (state) {
      for (int name = S_WINDSPEED; name <= S_PRESSURE; name++) {
         for (unsigned int id = 0;
            id < m_sensors.numberOfSensors((SensorName)name); id++) {

               if (!(tick % samplinRate((SensorName)name))) {
                                                   switch (name) {
                                                         case S_WINDSPEED:  A0(52); break;
                                                         case S_WINDDIRECTION: A0(51); break;
                                                         case S_TEMPERATURE: A0(53); break;
                                                         default:
                                                               // опрос давления и влажности произошел
                                                               // во время опроса температуры
                                                               break;
                                                   }
               }
         }
```

```
            if (!(tick%5)) {
                m_displayManager.displayDate(m_timeDate.currentDate());
                m_displayManager.displayTime(m_timeDate.currentTime());
            }
        }
    }
};

// Sensor.cpp:
// Класс датчиков
////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Interfaces.h"

Sensor::Sensor(SensorName sensorName, unsigned int id/* = 0*/) :
    m_name(sensorName),
    m_id(id)
{
            m_value = 0;
};

Sensor::~Sensor()
{
};

// Имя датчика
SensorName Sensor::name() const
{
    return m_name;
};

// ID датчика
unsigned int Sensor::id() const
{
    return m_id;
};

// Sensors.cpp
// Коллекция датчиков
////////////////////////////////////////////////////////////////////////

#include "stdafx.h"

// Шаблон класса коллекции
template <class C> Collection <C>::Collection():
m_nLen(10)
{
    for (int i=0; i<m_nLen; i++) {
        m_pCount[i] = 0;
    };
};

template <class C>  Collection <C>::~Collection() {
};

// добавление в коллекцию
template <class C> void Collection <C>::add(int x, C* item) {
    m_pArray[x][m_pCount[x]++] = item;
```

```
};

// возвращение значения зи коллекции
template <class C> C* Collection <C>::get(int x, int y) {
   return m_pArray[x][y];
};

// возвращение количества данных одного типа в коллекции
template <class C> int Collection <C>::getNumber(int x) {
   if ((x < 0)  || (x >= m_nLen)) return 0;
   return m_pCount[x];
};


Sensors::Sensors():
   Collection <Sensor>()
{
};

Sensors::~Sensors() {
};

// Добавление датчика в коллекцию
void Sensors::addSensor(Sensor& SensorName) {
   add(SensorName.name(), &SensorName);
};

// Возвращает количества датчиков в коллекции
unsigned int Sensors::numberOfSensors() {
   int count = 0;
         for (int i=0; i<100; i++) {
      count += getNumber(i);
         }
   return count;
};

// Возвращает количество датчиков определенного типа
unsigned int Sensors::numberOfSensors(SensorName sensorName) {
   return getNumber(sensorName);
};

// Возвращает датчик из коллекции
Sensor* Sensors::sensor(SensorName sensorName,unsigned int id/* = 0*/) {
   return get(sensorName, id);
};

// TemperatureSensor.cpp: implementation of the CInterfaces class.
// Поддержание информации о текущей температуры
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"

TemperatureSensor::TemperatureSensor(unsigned int id/* = 0*/) :
   TrendSensor(S_TEMPERATURE, id)
{
};

TemperatureSensor::~TemperatureSensor() {
};
```

```
// "сырое" значение температуры
float TemperatureSensor::rawValue() {
   m_value += (float)((1 - (rand() / 16384.0))/2);
            if (m_value < 0) m_value = 0;
            if (m_value > 1) m_value = 1;
            return m_value;//(float)(rand() / 32767.0);
};

// текущая температура
float TemperatureSensor::currentTemperature() {
   return currentValue();
};

// TimeDate.cpp
// Поддержка информации о текущем времени и о текущей дате
////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"

TimeDate::TimeDate() {
};

TimeDate::~TimeDate() {
};

// текущее время
const char *TimeDate::currentTime() {
   char *time = new char[9];
   GetLocalTime(&m_time);
   sprintf(time, "%02d:%02d:%02d", m_time.wHour, m_time.wMinute, m_time.wSecond);
   return time;
};

// текущая дата
const char *TimeDate::currentDate() {
   char *date = new char[11];
   GetLocalTime(&m_time);
   sprintf(date, "%02d-%02d-%d", m_time.wDay, m_time.wMonth, m_time.wYear);
   return date;
};


// установка часов
void TimeDate::setHour(WORD hour) {
            _SYSTEMTIME time;
   GetLocalTime(&time);
            m_time.wDay = time.wDay;
            m_time.wDayOfWeek = time.wDayOfWeek;
            m_time.wMonth = time.wMonth;
            m_time.wYear = time.wYear;

   m_time.wHour = hour;
   SetSystemTime(&m_time);
};

// установка минут
void TimeDate::setMinute(WORD min) {
            _SYSTEMTIME time;
```

```
    GetLocalTime(&time);
            m_time.wDay = time.wDay;
            m_time.wDayOfWeek = time.wDayOfWeek;
            m_time.wMonth = time.wMonth;
            m_time.wYear = time.wYear;

    m_time.wMinute = min;
    SetSystemTime(&m_time);
};

// установка секунд
void TimeDate::setSecond(WORD sec) {
            _SYSTEMTIME time;
    GetLocalTime(&time);
            m_time.wDay = time.wDay;
            m_time.wDayOfWeek = time.wDayOfWeek;
            m_time.wMonth = time.wMonth;
            m_time.wYear = time.wYear;

    SetSystemTime(&m_time);
};

// установка месяца
void TimeDate::setMonth(WORD  month) {
            _SYSTEMTIME time;
    GetLocalTime(&time);
            m_time.wHour = time.wHour;
            m_time.wMinute = time.wMinute;
            m_time.wSecond = time.wSecond;
            m_time.wMilliseconds = time.wMilliseconds;

    m_time.wMonth = month;
    SetSystemTime(&m_time);
};

// установка дня
void TimeDate::setDay(WORD  day) {
            _SYSTEMTIME time;
    GetLocalTime(&time);
            m_time.wHour = time.wHour;
            m_time.wMinute = time.wMinute;
            m_time.wSecond = time.wSecond;
            m_time.wMilliseconds = time.wMilliseconds;

    m_time.wDay = day;
    SetSystemTime(&m_time);
};

// установка года
void TimeDate::setYear(WORD  year) {
            _SYSTEMTIME time;
    GetLocalTime(&time);
            m_time.wHour = time.wHour;
            m_time.wMinute = time.wMinute;
            m_time.wSecond = time.wSecond;
            m_time.wMilliseconds = time.wMilliseconds;

    m_time.wYear = year;
    SetSystemTime(&m_time);
```

```
};

// Timer.cpp: implementation of the CInterfaces class.
// Обеспечение прерываний и диспетчеризация функций обратного вызова
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"

// установка функций обратоного вызова
void Timer::setCallback ( VOID (CALLBACK *func)(HWND, UINT, UINT, DWORD) ) {
   m_func = func;
};

// запуск таймера
void Timer::startTiming ( void ) {
   m_tickNumber = 0;
   SetTimer(NULL, NULL, 20, m_func);
};

// возвращает количество "тиков" таймера
unsigned int Timer::getNumberOfTicks () {
   return m_tickNumber++;
};

// TrendSensor.cpp
// Определение тренда давления или температуры как наклона графика
// (в линейном приближении) изменения их значений за данный интервал времени
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"

TrendSensor::TrendSensor(SensorName sensorName, unsigned int id/* = 0*/) :
   HistoricalSensor(sensorName, id)
{
}

TrendSensor::~TrendSensor() {
};

// тренд
float TrendSensor::trend() const {
   return m_trend;
};

// текущее значение
float TrendSensor::currentValue() {

          float currentValue = HistoricalSensor::currentValue();
          float prevValue = (currentValue - getLowValue())/(getHighValue()-getLowValue());

          m_trend = prevValue-m_previousValue;
   m_previousValue = prevValue;
   return currentValue;
};

// WindDirectionSensor.cpp: implementation of the CInterfaces class.
// Поддержание информации о текущем направлении ветра,
// указываемом как точка на розе ветров
//////////////////////////////////////////////////////////////////////
```

```
#include "stdafx.h"

WindDirectionSensor::WindDirectionSensor(unsigned int id/* = 0*/) :
   Sensor(S_WINDDIRECTION, id)
{
}

WindDirectionSensor::~WindDirectionSensor() {
};

// "сырое" значение направления ветра
float WindDirectionSensor::rawValue() {
   m_value += (float)((1 - (rand() / 16384.0))/8);
          if (m_value < -1) m_value = -1;
          if (m_value > 1) m_value = 1;
          return m_value;//(float)(rand() / 32767.0);
};

// текущее направление ветра
float WindDirectionSensor::currentValue() {
   float value = rawValue();
   value = (value*6.2832/*360-180*/);
   return value;
};

// текущее направление ветра
float WindDirectionSensor::currentWindDirection()
{
   return currentValue();
};

// WindSpeedSensor.cpp
// Поддержание информации о текущей скорости ветра
////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
WindSpeedSensor::WindSpeedSensor(unsigned int id/* = 0*/) :
   HistoricalSensor(S_WINDSPEED, id)
{
};

WindSpeedSensor::~WindSpeedSensor() {
};

// "сырое" значение скорости ветра
float WindSpeedSensor::rawValue() {
   m_value += (float)((1 - (rand() / 16384.0))/10);
          if (m_value < 0) m_value = 0;
          if (m_value > 1) m_value = 1;
          return m_value;//(float)(rand() / 32767.0);
};

// текущая скорость ветра
float WindSpeedSensor::currentWindSpeed() {
   return currentValue();
};

// Auto.h: класс автоматов.
```

```
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_AUTO_H__192031B2_DDE5_40BC_B30F_48F7ACFA19DB__INCLUDED_)
#define AFX_AUTO_H__192031B2_DDE5_40BC_B30F_48F7ACFA19DB__INCLUDED_

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

extern InputManager* inputManager;
extern DisplayManager *displayManager;
extern Sampler *sampler;
extern Timer *timer;
extern Keypad  *keypad;

extern int a_y0;
void A0(int e);

#endif // !defined(AFX_AUTO_H__192031B2_DDE5_40BC_B30F_48F7ACFA19DB__INCLUDED_)


// auto.cpp
// Класс автомататов
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "auto.h"

InputManager* inputManager;
DisplayManager *displayManager;
Sampler *sampler;
Timer *timer;
Keypad  *keypad;

TimeDate timeDate;

// Переменные
Sensors sensors;

TemperatureSensor   temperature;
PressureSensor      pressure;
HumiditySensor      humidity;
WindSpeedSensor     windSpeed;
WindDirectionSensor windDirection;

int a_y0 = 0, a_y1, a_y2 = 0, e;
char tbuffer[20];
char buf1[20], buf2[20];

#define LOG_ACTION 1
#define ACTIONS_LOGGING 1
#define GRAPH_EVENTS_LOGGING 1
#define GRAPH_TRANS_LOGGING 1
#define GRAPH_ENDS_LOGGING 1

void log_write(int act, char *descr, int numb) {
        FILE *f;
        f = fopen("graph.log", "a");
```

```
        fprintf(f, "%s* %s\n", timeDate.currentTime(), descr);
        fclose(f);
};

void log_exec(char *graph, int state, int acr) {
        FILE *f;
        f = fopen("graph.log", "a");
        fprintf(f, "%s{ %s: в состоянии %d запущен с событием e%02d\n", timeDate.currentTime(), graph, state, acr);
        fclose(f);
};

void log_trans(char *graph, int state_old, int state) {
        FILE *f;
        if (state_old == state) return;
        f = fopen("graph.log", "a");
        fprintf(f, "%sT %s: перешел из состояния %d  в состояние %d\n", timeDate.currentTime(), graph, state_old,
state);
        fclose(f);
};

void log_end(char *graph, int state, int acr) {
        FILE *f;
        f = fopen("graph.log", "a");
        fprintf(f, "%s} %s: завершил обработку события e%02d в состоянии %d\n", timeDate.currentTime(), graph, acr,
state);
        fclose(f);
};

void A1(int e);
void A2(int e);
void A3(int e);

////////////////////////////////////////////////////////////////////////
// Начались Автоматы
////////////////////////////////////////////////////////////////////////


//Инициализация
void z00() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z00. Инициализация", 0 ) ;
        #endif

        // Создаем сенсоры
        temperature.setHighValue(35);
        temperature.setLowValue(15);
        sensors.addSensor(temperature);

        pressure.setHighValue(80);
        pressure.setLowValue(30);
        sensors.addSensor(pressure);


        sensors.addSensor(humidity);
        humidity.setHighValue(100);
        humidity.setLowValue(20);

        sensors.addSensor(windSpeed);
        windSpeed.setHighValue(20);
```

```
        windSpeed.setLowValue(0);

        sensors.addSensor(windDirection);

        //
        keypad = new Keypad();
        inputManager = new InputManager(*keypad);

        // Задаем период опроса сенсоров
        sampler = new Sampler(sensors, *displayManager, *inputManager);
        sampler->setSamplinRate(S_WINDSPEED, 50);
        sampler->setSamplinRate(S_WINDDIRECTION, 100);
        sampler->setSamplinRate(S_TEMPERATURE, 200);
        sampler->setSamplinRate(S_HUMIDITY, 200);
        sampler->setSamplinRate(S_PRESSURE, 200);

    timer = new Timer();
};

//Опрос датчика направления ветра
void z06()  {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z06. Опрос датчика направления ветра", 0 ) ;
        #endif
        Sensor *sensor = sensors.sensor(S_WINDDIRECTION);
    displayManager->display(*sensor);

};

//Опрос датчика скорости ветра
void z07() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z07. Опрос датчика скорости ветра", 0 ) ;
        #endif
        Sensor *sensor = sensors.sensor(S_WINDSPEED);
    displayManager->display(*sensor);
};

//Опрос остальных датчиков
void z08() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z08. Опрос остальных датчиков", 0 ) ;
        #endif
        Sensor *sensor = sensors.sensor(S_TEMPERATURE);
    displayManager->display(*sensor);
        sensor = sensors.sensor(S_HUMIDITY);
    displayManager->display(*sensor);
        sensor = sensors.sensor(S_PRESSURE);
    displayManager->display(*sensor);

};

//Сообщение: Калибровка
void z03() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z03. Сообщение: Калибровка", 0 ) ;
        #endif

        displayManager->displayValueDate("", "");
```

```
        displayManager->displayValueTime("", "");
        displayManager->flashLabel(F_MODE);
        displayManager->displayMode("КАЛИБРОВКА");
        a_y2 = 0;
};

//Сообщение: Режим
void z05() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z05. Сообщение: Режим", 0 ) ;
        #endif

        displayManager->displayValueDate("", "");
        displayManager->displayValueTime("", "");
        displayManager->flashLabel(F_MODE);
        displayManager->displayMode("РЕЖИМ");
};

//Сообщение: Выбор
void z04() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z04. Сообщение: Выбор", 0 ) ;
        #endif

        displayManager->displayValueDate("", "");
        displayManager->displayValueTime("", "");
        displayManager->flashLabel(F_MODE);
        displayManager->displayMode("ВЫБОР");
};

//Очистить экран от предыдущих сообщений
void z02() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z02. Очистить экран от предыдущих сообщений", 0 ) ;
        #endif

        displayManager->flashLabel(F_OFF);
};

//Сообщение: Температура
void z09() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z09. Сообщение: Температура", 0 ) ;
        #endif

        displayManager->displayValueTemperature(INVISIBLE);
        displayManager->flashLabel(F_TEMPERATURE);
};

//Сообщение: Давление
void z10() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z10. Сообщение: Давление", 0 ) ;
        #endif

        displayManager->displayValuePressure(INVISIBLE);
        displayManager->flashLabel(F_PRESSURE);
};
```

```
//Сообщение: Влажность
void z11() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z11. Сообщение: Влажность", 0 ) ;
	#endif

	displayManager->displayValueHumidity(INVISIBLE);
	displayManager->flashLabel(F_HUMIDITY);
};

//Сообщение: Скорость ветра
void z12() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z12. Сообщение: Скорость ветра", 0 ) ;
	#endif

	displayManager->displayValueWind(INVISIBLE);
	displayManager->flashLabel(F_SPEED);
};

//Выбор максимального калибровочного значения температуры
void z27() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z27. Выбор максимального калибровочного значения температуры", 0 ) ;
	#endif

	a_y2 = 1;
	//displayManager->displayValueTemperature(inputManager->max_temp_value);
	TemperatureSensor *sensor = (TemperatureSensor *)sensors.sensor(S_TEMPERATURE);
	displayManager->displayValueTemperature(sensor->getHighValue());
};

//Выбор минимального калибровочного значения температуры
void z28() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z28. Выбор минимального калибровочного значения температуры", 0 ) ;
	#endif

	a_y2 = 2;
	TemperatureSensor *sensor = (TemperatureSensor *)sensors.sensor(S_TEMPERATURE);
	displayManager->displayValueTemperature(sensor->getLowValue());
};

//Уменьшение калибровочного значения температуры
void z29() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z29. Уменьшение калибровочного значения температуры", 0 ) ;
	#endif

	TemperatureSensor *sensor = (TemperatureSensor *)sensors.sensor(S_TEMPERATURE);
	if (a_y2 == 2) {
		sensor->setLowValue(sensor->getLowValue()-0.1f);
		displayManager->displayValueTemperature(sensor->getLowValue());
	}
	else {
		if (sensor->getHighValue() > sensor->getLowValue()) sensor->setHighValue(sensor-
>getHighValue()-0.1f);
		displayManager->displayValueTemperature(sensor->getHighValue());
	}
```

```
};

//Увеличение калибровочного значения температуры
void z30() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z30. Увеличение калибровочного значения температуры", 0 ) ;
	#endif

	TemperatureSensor *sensor = (TemperatureSensor *)sensors.sensor(S_TEMPERATURE);
	if (a_y2 == 2) {
		if (sensor->getLowValue() < sensor->getHighValue()) sensor->setLowValue(sensor-
>getLowValue()+0.1f);
		displayManager->displayValueTemperature(sensor->getLowValue());
	}
	else {
		sensor->setHighValue(sensor->getHighValue()+0.1f);
		displayManager->displayValueTemperature(sensor->getHighValue());
	}
};

//Выбор максимального калибровочного значения давления
void z31() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z31. Выбор максимального калибровочного значения давления", 0 ) ;
	#endif

	PressureSensor *sensor = (PressureSensor *)sensors.sensor(S_PRESSURE);
	a_y2 = 1;
	displayManager->displayValueTemperature(sensor->getHighValue());
};

//Выбор минимального калибровочного значения давления
void z32() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z32. Выбор минимального калибровочного значения давления", 0 ) ;
	#endif

	PressureSensor *sensor = (PressureSensor *)sensors.sensor(S_PRESSURE);
	a_y2 = 2;
	displayManager->displayValueTemperature(sensor->getLowValue());
};

//Уменьшение калибровочного значения давления
void z33() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z33. Уменьшение калибровочного значения давления", 0 ) ;
	#endif

	PressureSensor *sensor = (PressureSensor *)sensors.sensor(S_PRESSURE);
	if (a_y2 == 2) {
		if (sensor->getLowValue() > 1) sensor->setLowValue(sensor->getLowValue() - 1);
		displayManager->displayValuePressure(sensor->getLowValue());
	}
	else {
		if (sensor->getHighValue() > sensor->getLowValue() ) sensor->setHighValue(sensor-
>getHighValue() - 1);
		displayManager->displayValuePressure(sensor->getHighValue());
	}
};
```

```
//Увеличение калибровочного значения давления
void z34() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z34. Увеличение калибровочного значения давления", 0 ) ;
	#endif

	PressureSensor *sensor = (PressureSensor *)sensors.sensor(S_PRESSURE);
	if (a_y2 == 2) {
		if (sensor->getLowValue() < sensor->getHighValue()) sensor->setLowValue(sensor->getLowValue()
+ 1);
		displayManager->displayValuePressure(sensor->getLowValue());
	}
	else {
		sensor->setHighValue(sensor->getHighValue() + 1);
		displayManager->displayValuePressure(sensor->getHighValue());
	}
};

//Выбор максимального калибровочного значения влажности
void z35() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z35. Выбор максимального калибровочного значения влажности", 0 ) ;
	#endif

	HumiditySensor *sensor = (HumiditySensor *)sensors.sensor(S_HUMIDITY);
	a_y2 = 1;
	displayManager->displayValueTemperature(sensor->getHighValue());
};

//Выбор минимального калибровочного значения влажности
void z36() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z36. Выбор минимального калибровочного значения влажности", 0 ) ;
	#endif

	HumiditySensor *sensor = (HumiditySensor *)sensors.sensor(S_HUMIDITY);
	a_y2 = 2;
	displayManager->displayValueTemperature(sensor->getLowValue());
};

//Уменьшение калибровочного значения влажности
void z37() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z37. Уменьшение калибровочного значения влажности", 0 ) ;
	#endif

	HumiditySensor *sensor = (HumiditySensor *)sensors.sensor(S_HUMIDITY);
	if (a_y2 == 2) {
		if (sensor->getLowValue() > 1) sensor->setLowValue(sensor->getLowValue() - 1);
		displayManager->displayValueHumidity(sensor->getLowValue());
	}
	else {
		if (sensor->getHighValue() > sensor->getLowValue() ) sensor->setHighValue(sensor-
>getHighValue() - 1);
		displayManager->displayValueHumidity(sensor->getHighValue());
	}
};
```

```
//Увеличение калибровочного значения влажности
void z38() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z38. Увеличение калибровочного значения влажности", 0 ) ;
	#endif

	HumiditySensor *sensor = (HumiditySensor *)sensors.sensor(S_HUMIDITY);
	if (a_y2 == 2) {
		if (sensor->getLowValue() < sensor->getHighValue()) sensor->setLowValue(sensor->getLowValue()
+ 1);
		displayManager->displayValueHumidity(sensor->getLowValue());
	}
	else {
		sensor->setHighValue(sensor->getHighValue() + 1);
		displayManager->displayValueHumidity(sensor->getHighValue());
	}
};

//Выбор максимального калибровочного значения скорости ветра
void z39() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z39. Выбор максимального калибровочного значения скорости ветра", 0 )
;
	#endif

	WindSpeedSensor *sensor = (WindSpeedSensor *)sensors.sensor(S_WINDSPEED);
	a_y2 = 1;
	displayManager->displayValueTemperature(sensor->getHighValue());
};

//Выбор минимального калибровочного значения скорости ветра
void z40() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z40. Выбор минимального калибровочного значения скорости ветра", 0 )
;
	#endif

	WindSpeedSensor *sensor = (WindSpeedSensor *)sensors.sensor(S_WINDSPEED);
	a_y2 = 2;
	displayManager->displayValueTemperature(sensor->getLowValue());
};

//Уменьшение калибровочного значения скорости ветра
void z41() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z41. Уменьшение калибровочного значения скорости ветра", 0 ) ;
	#endif

	WindSpeedSensor *sensor = (WindSpeedSensor *)sensors.sensor(S_WINDSPEED);
	if (a_y2 == 2) {
		if (sensor->getLowValue() > 0.1f) sensor->setLowValue(sensor->getLowValue() - 0.1f);
		displayManager->displayValueWind(sensor->getLowValue());
	}
	else {
		if ((sensor->getHighValue() > 0.1f) && (sensor->getHighValue() > sensor->getLowValue() )) sensor-
>setHighValue(sensor->getHighValue() - 0.1f);
		displayManager->displayValueWind(sensor->getHighValue());
	}
};
```

```
//Увеличение калибровочного значения скорости ветра
void z42() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z42. Увеличение калибровочного значения скорости ветра", 0 ) ;
	#endif

	WindSpeedSensor *sensor = (WindSpeedSensor *)sensors.sensor(S_WINDSPEED);
	if (a_y2 == 2) {
		if (sensor->getLowValue() < sensor->getHighValue()) sensor->setLowValue(sensor->getLowValue()
+ 0.1f);
		displayManager->displayValueWind(sensor->getLowValue());
	}
	else {
		sensor->setHighValue(sensor->getHighValue() + 0.1f);
		displayManager->displayValueWind(sensor->getHighValue());
	}
};

//Сообщение: Время
void z13() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z13. Сообщение: Время", 0 ) ;
	#endif

	displayManager->flashLabel(F_TIME);
	strcpy(&buf2[2],timeDate.currentTime());
	FillMemory(buf2, 4, 32);
	displayManager->displayValueTime(timeDate.currentTime(), buf2);
	a_y2 = 0;
};

//Сообщение: Дата
void z14() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z14. Сообщение: Дата", 0 ) ;
	#endif

	displayManager->flashLabel(F_DATE);
	strcpy(&buf2[2],timeDate.currentDate());
	FillMemory(buf2, 4, 32);
	displayManager->displayValueDate(timeDate.currentDate(), buf2);
	a_y2 = 0;
};

//Вывод максимального значения температуры
void z15() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z15. Вывод максимального значения температуры", 0 ) ;
	#endif

	TemperatureSensor *sensor = (TemperatureSensor *)sensors.sensor(S_TEMPERATURE);
	displayManager->displayValueTemperature(sensor->highValue());
	CopyMemory(tbuffer, sensor->timeOfHighValue(), 19);
	displayManager->displayValueDate(&tbuffer[9], &tbuffer[9]);
	displayManager->displayValueTime(tbuffer, tbuffer);
};

//Вывод минимального значения температуры
```

```
void z16() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z16. Вывод минимального значения температуры", 0 ) ;
	#endif

	TemperatureSensor *sensor = (TemperatureSensor *)sensors.sensor(S_TEMPERATURE);
	displayManager->displayValueTemperature(sensor->lowValue());
	CopyMemory(tbuffer, sensor->timeOfLowValue(), 19);
	displayManager->displayValueDate(&tbuffer[9], &tbuffer[9]);
	displayManager->displayValueTime(tbuffer, tbuffer);
};

//Вывод максимального значения давления
void z17() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z17. Вывод максимального значения давления", 0 ) ;
	#endif

	PressureSensor *sensor = (PressureSensor *)sensors.sensor(S_PRESSURE);
	displayManager->displayValuePressure(sensor->highValue());
	CopyMemory(tbuffer, sensor->timeOfHighValue(), 19);
	displayManager->displayValueDate(&tbuffer[9], &tbuffer[9]);
	displayManager->displayValueTime(tbuffer, tbuffer);
};

//Вывод минимального значения давления
void z18() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z18. Вывод минимального значения давления", 0 ) ;
	#endif

	PressureSensor *sensor = (PressureSensor *)sensors.sensor(S_PRESSURE);
	displayManager->displayValuePressure(sensor->lowValue());
	CopyMemory(tbuffer, sensor->timeOfLowValue(), 19);
	displayManager->displayValueDate(&tbuffer[9], &tbuffer[9]);
	displayManager->displayValueTime(tbuffer, tbuffer);
};

//Вывод максимального значения влажности
void z19() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z19. Вывод максимального значения влажности", 0 ) ;
	#endif

	HumiditySensor *sensor = (HumiditySensor *)sensors.sensor(S_HUMIDITY);
	displayManager->displayValueHumidity(sensor->highValue());
	CopyMemory(tbuffer, sensor->timeOfHighValue(), 19);
	displayManager->displayValueDate(&tbuffer[9], &tbuffer[9]);
	displayManager->displayValueTime(tbuffer, tbuffer);
};

//Вывод минимального значения влажности
void z20() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z20. Вывод минимального значения влажности", 0 ) ;
	#endif

	HumiditySensor *sensor = (HumiditySensor *)sensors.sensor(S_HUMIDITY);
	displayManager->displayValueHumidity(sensor->lowValue());
```

```
        CopyMemory(tbuffer, sensor->timeOfLowValue(), 19);
        displayManager->displayValueDate(&tbuffer[9], &tbuffer[9]);
        displayManager->displayValueTime(tbuffer, tbuffer);
};

//Вывод максимального значения скорости ветра
void z21() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z21. Вывод максимального значения скорости ветра", 0 ) ;
        #endif

        WindSpeedSensor *sensor = (WindSpeedSensor *)sensors.sensor(S_WINDSPEED);
        displayManager->displayValueWind(sensor->highValue());
        CopyMemory(tbuffer, sensor->timeOfHighValue(), 19);
        displayManager->displayValueDate(&tbuffer[9], &tbuffer[9]);
        displayManager->displayValueTime(tbuffer, tbuffer);
};

//Вывод минимального значения скорости ветра
void z22() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z22. Вывод минимального значения скорости ветра", 0 ) ;
        #endif

        WindSpeedSensor *sensor = (WindSpeedSensor *)sensors.sensor(S_WINDSPEED);
        displayManager->displayValueWind(sensor->lowValue());
        CopyMemory(tbuffer, sensor->timeOfLowValue(), 19);
        displayManager->displayValueDate(&tbuffer[9], &tbuffer[9]);
        displayManager->displayValueTime(tbuffer, tbuffer);
};

//Увеличение значения поля времени
void z47() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z47. Увеличение значения поля времени", 0 ) ;
        #endif

        if (a_y2 == 0) {timeDate.setHour( (timeDate.m_time.wHour == 23) ? 0 : timeDate.m_time.wHour+1 );}
        else if (a_y2 == 1) {timeDate.setMinute ( (timeDate.m_time.wMinute == 59) ? 0 : timeDate.m_time.wMinute+1
);}
        else {timeDate.setSecond( (timeDate.m_time.wSecond == 59) ? 0 : timeDate.m_time.wSecond+1 );}

        sprintf(buf1, "%02d:%02d:%02d", timeDate.m_time.wHour, timeDate.m_time.wMinute,
timeDate.m_time.wSecond);
        if (a_y2 == 0) sprintf(buf2, "    :%02d:%02d", timeDate.m_time.wMinute, timeDate.m_time.wSecond);
        else if (a_y2 == 1) sprintf(buf2, "%02d:    :%02d", timeDate.m_time.wHour, timeDate.m_time.wSecond);
        else sprintf(buf2, "%02d:%02d:    ", timeDate.m_time.wHour, timeDate.m_time.wMinute);
        displayManager->displayValueTime(buf1, buf2);
};

//Уменьшение значения поля времени
void z48() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z48. Уменьшение значения поля времени", 0 ) ;
        #endif

        if (a_y2 == 0) {timeDate.setHour( (timeDate.m_time.wHour == 0) ? 23 : timeDate.m_time.wHour-1 );}
        else if (a_y2 == 1) {timeDate.setMinute( (timeDate.m_time.wMinute == 0) ? 59 : timeDate.m_time.wMinute-1
);}
```

```
        else {timeDate.setSecond( (timeDate.m_time.wSecond == 0) ? 49 : timeDate.m_time.wSecond-1 );}

        sprintf(buf1, "%02d:%02d:%02d", timeDate.m_time.wHour, timeDate.m_time.wMinute,
timeDate.m_time.wSecond);
        if (a_y2 == 0) sprintf(buf2, "    :%02d:%02d", timeDate.m_time.wMinute, timeDate.m_time.wSecond);
        else if (a_y2 == 1) sprintf(buf2, "%02d:    :%02d", timeDate.m_time.wHour, timeDate.m_time.wSecond);
        else sprintf(buf2, "%02d:%02d:    ", timeDate.m_time.wHour, timeDate.m_time.wMinute);
        displayManager->displayValueTime(buf1, buf2);
};

//Переход к предыдущему полю времени
void z49() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z49. Переход к предыдущему полю времени", 0 ) ;
        #endif

        a_y2 = (a_y2 == 0) ? 2 : a_y2-1;
        sprintf(buf1, "%02d:%02d:%02d", timeDate.m_time.wHour, timeDate.m_time.wMinute,
timeDate.m_time.wSecond);
        if (a_y2 == 0) sprintf(buf2, "    :%02d:%02d", timeDate.m_time.wMinute, timeDate.m_time.wSecond);
        else if (a_y2 == 1) sprintf(buf2, "%02d:    :%02d", timeDate.m_time.wHour, timeDate.m_time.wSecond);
        else sprintf(buf2, "%02d:%02d:    ", timeDate.m_time.wHour, timeDate.m_time.wMinute);
        displayManager->displayValueTime(buf1, buf2);
};

//Переход к следующему полю времени
void z50() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z50. Переход к следующему полю времени", 0 ) ;
        #endif

        a_y2 = (a_y2 == 2) ? 0 : a_y2+1;
        sprintf(buf1, "%02d:%02d:%02d", timeDate.m_time.wHour, timeDate.m_time.wMinute,
timeDate.m_time.wSecond);
        if (a_y2 == 0) sprintf(buf2, "    :%02d:%02d", timeDate.m_time.wMinute, timeDate.m_time.wSecond);
        else if (a_y2 == 1) sprintf(buf2, "%02d:    :%02d", timeDate.m_time.wHour, timeDate.m_time.wSecond);
        else sprintf(buf2, "%02d:%02d:    ", timeDate.m_time.wHour, timeDate.m_time.wMinute);
        displayManager->displayValueTime(buf1, buf2);

};

//Увеличение значения поля даты
void z43() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z43. Увеличение значения поля даты", 0 ) ;
        #endif

        if (a_y2 == 0) {timeDate.setDay( (timeDate.m_time.wDay == 31) ? 1 : timeDate.m_time.wDay+1 );}
        else if (a_y2 == 1) {timeDate.setMonth( (timeDate.m_time.wMonth == 12) ? 1 : timeDate.m_time.wMonth+1 );}
        else {timeDate.setYear( (timeDate.m_time.wYear == 2080) ? 1980 : timeDate.m_time.wYear+1 );}

        sprintf(buf1, "%02d-%02d-%d", timeDate.m_time.wDay, timeDate.m_time.wMonth, timeDate.m_time.wYear);
        if (a_y2 == 0) sprintf(buf2, "    -%02d-%d", timeDate.m_time.wMonth, timeDate.m_time.wYear);
        else if (a_y2 == 1) sprintf(buf2, "%02d-    -%d", timeDate.m_time.wDay, timeDate.m_time.wYear);
        else sprintf(buf2, "%02d-%02d-    ", timeDate.m_time.wDay, timeDate.m_time.wMonth);
        displayManager->displayValueDate(buf1, buf2);
};

//Уменьшение значения поля даты
```

```
void z44() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z44. Уменьшение значения поля даты", 0 ) ;
	#endif

	if (a_y2 == 0) {timeDate.setDay( (timeDate.m_time.wDay == 1) ? 31 : timeDate.m_time.wDay-1 );}
	else if (a_y2 == 1) {timeDate.setMonth( (timeDate.m_time.wMonth == 1) ? 12 : timeDate.m_time.wMonth-1 );}
	else {timeDate.setYear( (timeDate.m_time.wYear == 1980) ? 2080 : timeDate.m_time.wYear-1 );}

	sprintf(buf1, "%02d-%02d-%d", timeDate.m_time.wDay, timeDate.m_time.wMonth, timeDate.m_time.wYear);
	if (a_y2 == 0) sprintf(buf2, "    -%02d-%d", timeDate.m_time.wMonth, timeDate.m_time.wYear);
	else if (a_y2 == 1) sprintf(buf2, "%02d-    -%d", timeDate.m_time.wDay, timeDate.m_time.wYear);
	else sprintf(buf2, "%02d-%02d-    ", timeDate.m_time.wDay, timeDate.m_time.wMonth);
	displayManager->displayValueDate(buf1, buf2);
};

//Переход к предыдущему полю даты
void z45() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z45. Переход к предыдущему полю даты", 0 ) ;
	#endif

	a_y2 = (a_y2 == 0) ? 2 : a_y2-1;
	sprintf(buf1, "%02d-%02d-%d", timeDate.m_time.wDay, timeDate.m_time.wMonth, timeDate.m_time.wYear);
	if (a_y2 == 0) sprintf(buf2, "    -%02d-%d", timeDate.m_time.wMonth, timeDate.m_time.wYear);
	else if (a_y2 == 1) sprintf(buf2, "%02d-    -%d", timeDate.m_time.wDay, timeDate.m_time.wYear);
	else sprintf(buf2, "%02d-%02d-    ", timeDate.m_time.wDay, timeDate.m_time.wMonth);
	displayManager->displayValueDate(buf1, buf2);
};

//Переход к следующему полю даты
void z46() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z46. Переход к следующему полю даты", 0 ) ;
	#endif

	a_y2 = (a_y2 == 2) ? 0 : a_y2+1;
	sprintf(buf1, "%02d-%02d-%d", timeDate.m_time.wDay, timeDate.m_time.wMonth, timeDate.m_time.wYear);
	if (a_y2 == 0) sprintf(buf2, "    -%02d-%d", timeDate.m_time.wMonth, timeDate.m_time.wYear);
	else if (a_y2 == 1) sprintf(buf2, "%02d-    -%d", timeDate.m_time.wDay, timeDate.m_time.wYear);
	else sprintf(buf2, "%02d-%02d-    ", timeDate.m_time.wDay, timeDate.m_time.wMonth);
	displayManager->displayValueDate(buf1, buf2);
};

//Установить следующую единицу измерения скорости ветра
void z23() {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z23. Установить следующую единицу измерения скорости ветра", 0 ) ;
	#endif

	a_y2 = (++a_y2)&1;
	displayManager->setSpeedScale(SS_KMH);
};

//Установить предыдущую единицу измерения скорости ветра
void z24()  {
	#ifdef ACTIONS_LOGGING
		log_write( LOG_ACTION, "z24. Установить предыдущую единицу измерения скорости ветра", 0 ) ;
	#endif
```

```
        a_y2 = (++a_y2)&1;
        displayManager->setSpeedScale(SS_KMH);
};

//Установить следующую единицу измерения температуры
void z25() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z25. Установить следующую единицу измерения температуры", 0 ) ;
        #endif

        displayManager->setTemperatureScale(TS_CELSIUS);
};
//Установить предыдущую единицу измерения температуры
void z26() {
        #ifdef ACTIONS_LOGGING
                log_write( LOG_ACTION, "z26. Установить предыдущую единицу измерения температуры", 0 ) ;
        #endif

        displayManager->setTemperatureScale(TS_CELSIUS);
};

void A0(int e) {
        #ifdef GRAPH_EVENTS_LOGGING
                log_exec( "A0", a_y0, e ) ;
        #endif

        int y_old = a_y0;
        switch (a_y0)    {
        case 0:
                z00();
                a_y0 = 1;
                break;
        case 1:
                if (e == 51) {z06();}
                else if (e == 52) {z07();}
                else if (e == 53) {z08();}
                else if (e == 41) {z03(); a_y0 = 2; A1(50);}
                else if (e == 40) {z04(); a_y0 = 3; A1(50);}
                else if (e == 42) {z05(); a_y0 = 4; A1(50);}
    break;
        case 2:
                A1(e);
                if (e == 11) {z02(); a_y0 = 1;}
                break;
        case 3:
                A2(e);
                if (e == 11) {z02(); a_y0 = 1;}
                break;
        case 4:
                A3(e);
                if (e == 11) {z02(); a_y0 = 1;}
                break;
        default:
                break;
  };

        #ifdef GRAPH_TRANS_LOGGING
                log_trans( "A0", y_old, a_y0 ) ;
```

```
	#endif

	#ifdef GRAPH_ENDS_LOGGING
		log_end( "A0", a_y0, e ) ;
	#endif
};

void A1(int e) {
	#ifdef GRAPH_EVENTS_LOGGING
		log_exec( "A1", a_y1, e ) ;
	#endif

	int y_old = a_y1;
	if (e == 50) {a_y1 = 0; return;}
	else if (e == 00) {z09(); a_y1 = 1; return;}
	else if (e == 02) {z10(); a_y1 = 2; return;}
	else if (e == 20) {z11(); a_y1 = 3; return;}
	else if (e == 22) {z12(); a_y1 = 4; return;}

	switch(a_y1)	{
	case 0:
		break;
	case 1:
		if (e == 01) {z27();}
		else if (e == 21) {z28();}
		else if (e == 10) {z29();}
		else if (e == 12) {z30();}
		break;
	case 2:
		if (e == 01) {z31();}
		else if (e == 21) {z32();}
		else if (e == 10) {z33();}
		else if (e == 12) {z34();}
		break;
	case 3:
		if (e == 01) {z35();}
		else if (e == 21) {z36();}
		else if (e == 10) {z37();}
		else if (e == 12) {z38();}
		break;
	case 4:
		if (e == 01) {z39();}
		else if (e == 21) {z40();}
		else if (e == 10) {z41();}
		else if (e == 12) {z42();}
		break;
	default:
		break;
	}

	#ifdef GRAPH_TRANS_LOGGING
		log_trans( "A1", y_old, a_y1 ) ;
	#endif

	#ifdef GRAPH_ENDS_LOGGING
		log_end( "A1", a_y1, e ) ;
	#endif

}
```

```
void A2(int e) {
	#ifdef GRAPH_EVENTS_LOGGING
		log_exec( "A2", a_y1, e ) ;
	#endif

	int y_old = a_y1;
	if (e == 50) {a_y1 = 0; return;}
	else if (e == 00) {z09(); a_y1 = 1; return;}
	else if (e == 02) {z10(); a_y1 = 2; return;}
	else if (e == 20) {z11(); a_y1 = 3; return;}
	else if (e == 22) {z12(); a_y1 = 4; return;}
	else if (e == 30) {z13(); a_y1 = 5; return;}
	else if (e == 32) {z14(); a_y1 = 6; return;}

	switch(a_y1) {
	case 0:
		break;
	case 1:
		if (e == 01) {z15();}
		else if (e == 21) {z16();}
		break;
	case 2:
		if (e == 01) {z17();}
		else if (e == 21) {z18();}
		break;
	case 3:
		if (e == 01) {z19();}
		else if (e == 21) {z20();}
		break;
	case 4:
		if (e == 01) {z21();}
		else if (e == 21) {z22();}
		break;
	case 5:
		if (e == 01) {z47();}
		else if (e == 21) {z48();}
		else if (e == 10) {z49();}
		else if (e == 12) {z50();}
		break;
	case 6:
		if (e == 01) {z43();}
		else if (e == 21) {z44();}
		else if (e == 10) {z45();}
		else if (e == 12) {z46();}
		break;
	default:
		break;
	}

	#ifdef GRAPH_TRANS_LOGGING
		log_trans( "A2", y_old, a_y1 ) ;
	#endif

	#ifdef GRAPH_ENDS_LOGGING
		log_end( "A2", a_y1, e ) ;
	#endif
}
```

```
void A3(int e) {
	#ifdef GRAPH_EVENTS_LOGGING
		log_exec( "A3", a_y1, e ) ;
	#endif

	int y_old = a_y1;
	if (e == 50) {a_y1 = 0; return;}
	else if (e == 22) {z12(); a_y1 = 1; return;}
	else if (e == 00) {z09(); a_y1 = 2; return;}

	switch(a_y1) {
	case 0:
		break;
	case 1:
		if (e == 01) {z23();}
		else if (e == 21) {z24();}
		break;
	case 2:
		if (e == 01) {z25();}
		else if (e == 21) {z26();}
	default:
		break;
	}

	#ifdef GRAPH_TRANS_LOGGING
		log_trans( "A3", y_old, a_y1 ) ;
	#endif

	#ifdef GRAPH_ENDS_LOGGING
		log_end( "A3", a_y1, e ) ;
	#endif
};

//////////////////////////////////////////////////////////////////////////
// Закончились Автоматы
//////////////////////////////////////////////////////////////////////////

#if !defined(AFX_METEO_H__56064E23_553A_43C8_A39D_C9DA477C5DA4__INCLUDED_)
#define AFX_METEO_H__56064E23_553A_43C8_A39D_C9DA477C5DA4__INCLUDED_

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

#include "resource.h"


#endif // !defined(AFX_METEO_H__56064E23_553A_43C8_A39D_C9DA477C5DA4__INCLUDED_)

// Meteo.cpp
// Главный модуль
//////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "resource.h"
#include <stdlib.h>
#include <time.h>
#include "auto.h"
```

```
#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];            // The title bar text

// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    KeyBoard(HWND, UINT, WPARAM, LPARAM);
int main();

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_METEO, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }

        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_METEO);

        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
        }

        return msg.wParam;
}

ATOM MyRegisterClass(HINSTANCE hInstance) {
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProc;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra         = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_METEO);
```

```
	wcex.hCursor		= LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground	= (HBRUSH)(COLOR_WINDOW+2);
	wcex.lpszMenuName	= NULL;
	wcex.lpszClassName	= szWindowClass;
	wcex.hIconSm		= LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

	return RegisterClassEx(&wcex);
}

HWND phWnd, phDlg = NULL;


BOOL InitInstance(HINSTANCE hInstance, int nCmdShow) {
	hInst = hInstance; // Store instance handle in our global variable
	phWnd = CreateWindow(szWindowClass, szTitle, WS_CAPTION|WS_SYSMENU,
		50, 50, 380, 192, NULL, NULL, hInstance, NULL);
	if (!phWnd) {
		return FALSE;
	}

	ShowWindow(phWnd, nCmdShow);
	UpdateWindow(phWnd);
	DialogBox(hInst, (LPCTSTR)IDD_KEYBOARD, NULL, (DLGPROC)KeyBoard);
	return TRUE;
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Processes messages for the main window.
//
//  WM_COMMAND	- process the application menu
//  WM_PAINT	- Paint the main window
//  WM_DESTROY	- post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	int wmId, wmEvent;
	PAINTSTRUCT ps;
	HDC hdc_main;
	TCHAR szHello[MAX_LOADSTRING];
	LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

	switch (message) {
	case WM_CREATE:
		phWnd = hWnd;
		// Выполняем главную процедуру
		main();
		break;
	case WM_COMMAND:
		wmId    = LOWORD(wParam);
		wmEvent = HIWORD(wParam);
		// Parse the menu selections:
		switch (wmId) {
		case IDM_EXIT:
			DestroyWindow(hWnd);
			break;
		default:
```

```
                return DefWindowProc(hWnd, message, wParam, lParam);
            }
            break;
        case WM_MOVING:
            if (phDlg != NULL) {
                LPRECT rc = (LPRECT)lParam;
                SetWindowPos(phDlg, HWND_TOP, rc->right, rc->top, 0, 0, SWP_NOSIZE);
            }
            break;
        case WM_PAINT:
            {
            RECT rt;
            GetClientRect(hWnd, &rt);
            hdc_main = BeginPaint(hWnd, &ps);
            HDC hdcMem = CreateCompatibleDC(hdc_main);
            HBITMAP hbmMem = CreateCompatibleBitmap(hdc_main, rt.right-rt.left, rt.bottom-rt.top);
            HGDIOBJ hOld   = SelectObject(hdcMem, hbmMem);

            displayManager->refresh(hdcMem);
            // Выводим построенное  изображение и памяти на экран
            BitBlt(hdc_main, 0, 0, rt.right-rt.left, rt.bottom-rt.top, hdcMem, 0, 0, SRCCOPY);

            // Освобождаем память
            SelectObject(hdcMem, hOld);
            DeleteObject(hbmMem);
            DeleteDC    (hdcMem);

            EndPaint(hWnd, &ps);
            }
            break;
        case WM_ERASEBKGND:
            return 1;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

// Виртуальная клавиатура
LRESULT CALLBACK KeyBoard(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message) {
        case WM_INITDIALOG:
            phDlg = hDlg;
            SetFocus(GetDlgItem(hDlg, IDC_STATIC));
            SetWindowPos(hDlg, HWND_TOP, 430, 50, 0, 0, SWP_NOSIZE);
            return TRUE;
        case WM_MOVING:
            if (phWnd != NULL) {
                LPRECT rc = (LPRECT)lParam;
                SetWindowPos(phWnd, HWND_TOP, rc->left-380, rc->top, 0, 0, SWP_NOSIZE);
            }
            break;
        case WM_COMMAND:
            SetFocus(GetDlgItem(hDlg, IDC_STATIC));
            switch (LOWORD(wParam)) {
```

```
                    case IDC_BUTTONTEMP:
                            keypad->keyPress(K_TEMP);
                            break;
                    case IDC_BUTTONUP:
                            keypad->keyPress(K_UP);
                            break;
                    case IDC_BUTTONPRESSURE:
                            keypad->keyPress(K_PRESSURE);
                            break;
                    case IDC_BUTTONLEFT:
                            keypad->keyPress(K_LEFT);
                            break;
                    case IDC_BUTTONRUN:
                            keypad->keyPress(K_RUN);
                            break;
                    case IDC_BUTTONRIGHT:
                            keypad->keyPress(K_RIGHT);
                            break;
                    case IDC_BUTTONHUMIDITY:
                            keypad->keyPress(K_HUMIDITY);
                            break;
                    case IDC_BUTTONDOWN:
                            keypad->keyPress(K_DOWN);
                            break;
                    case IDC_BUTTONWIND:
                            keypad->keyPress(K_WIND);
                            break;
                    case IDC_BUTTONTIME:
                            keypad->keyPress(K_TIME);
                            break;
                    case IDC_BUTTONDATE:
                            keypad->keyPress(K_DATE);
                            break;
                    case IDC_BUTTONSELECT:
                            keypad->keyPress(K_SELECT);
                            break;
                    case IDC_BUTTONCALIBRATE:
                            keypad->keyPress(K_CALIBRATE);
                            break;
                    case IDC_BUTTONMODE:
                            keypad->keyPress(K_MODE);
                            break;
                    default:
                            break;
                    }
            case WM_ACTIVATE:
                    SetFocus(GetDlgItem(hDlg, IDC_STATIC));
        break;
            }
    return FALSE;
}


VOID __stdcall acquire(HWND hwnd,  UINT uMsg, UINT idEvent,  DWORD dwTime) {
    sampler->sample(timer->getNumberOfTicks());
};


// Основная часть программы
```

```
int main ( void ) {
	srand( (unsigned)time( NULL ) );

	// Инициализация дисплея
	displayManager = new DisplayManager(phWnd);

	// Инициализация автомата
	A0(42);

	// Запускаем таймер
	timer->setCallback ( &acquire ) ;
   timer->startTiming () ;

	return 0;
};

#if !defined(AFX_STDAFX_H__A9DB83DB_A9FD_11D0_BFD1_444553540000__INCLUDED_)
#define AFX_STDAFX_H__A9DB83DB_A9FD_11D0_BFD1_444553540000__INCLUDED_

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

#define WIN32_LEAN_AND_MEAN		// Exclude rarely-used stuff from Windows headers

// Windows Header Files:
#include <windows.h>

// C RunTime Header Files
#include <stdlib.h>
#include <malloc.h>
#include <memory.h>
#include <tchar.h>

// Local Header Files
#include <stdio.h>
#include <Math.h>
#include "Interfaces.h"

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

#endif // !defined(AFX_STDAFX_H__A9DB83DB_A9FD_11D0_BFD1_444553540000__INCLUDED_)

// stdafx.cpp : source file that includes just the standard includes
//	Meteo.pch will be the pre-compiled header
//	stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```