Министерство образования и науки Российской Федерации
Балтийский государственный технический университет «Военмех»
Кафедра информационных систем и компьютерных технологий

ПРОГРАММИРОВАНИЕ НА ЯЗЫКАХ ВЫСОКОГО УРОВНЯ С ИСПОЛЬЗОВАНИЕМ WIN32 API

Методические указания
к выполнению курсовой работы

Санкт-Петербург
2008

УДК 004.43 (076)

Составитель *А.Н.Гущин*, канд. техн. наук.

Программирование на языках высокого уровня с использованием Win32 API: методические указания к выполнению курсовой работы / Сост. А.Н.Гущин; Балт. гос. техн. ун-т. – СПб., 2007. – с.

Методические указания к выполнению курсовой работы по дисциплине «Программирование на языках высокого уровня» содержат формулировку цели и содержания работы, краткие сведения из теории, последовательность этапов выполнения курсовой работы, требования к оформлению пояснительной записки, образцы тем курсовых работ, контрольные вопросы, а также перечень рекомендуемой литературы.

Предназначены для студентов второго курса, обучающихся по специальности «Автоматизированные системы обработки информации и управления».

Р е ц е н з е н т канд. техн. наук, доц. каф И5 БГТУ *В.Н. Уткин*

Утверждено
редакционно-издательским
советом университета

Цель и содержание работы

Цель работы – научить основным принципам разработки приложений для современных операционных систем, управляемых сообщениями, с использованием языков программирования высокого уровня, основанных на продукционной и объектно-ориентированной парадигмах программирования.

В процессе выполнения курсовой работы необходимо разработать приложение для операционной системы Windows XP, имеющее функциональность согласно индивидуальному заданию. Приложение должно иметь оконный пользовательский интерфейс, либо многооконный (несколько однотипных окон), либо однооконный (но не менее трех разнотипных окон). Для реализации необходимо использовать языки программирования C (взаимодействие с операционной системой) и C++ (реализация основной функциональности приложения). Для взаимодействия с операционной системой необходимо использовать непосредственные вызовы библиотек интерфейса прикладного программирования Win32 API, использование дополнительных сторонних библиотек-оберток (включая MFC) запрещается. Допускается создание разработчиком собственных объектных библиотек-оберток. При реализации функциональной части приложения должно использоваться не менее 4-х классов, расположенных не менее, чем в двух уровнях иерархии.

Для успешного выполнения курсовой работы необходимо знать и уметь использовать на практике при разработке программ:

— языки программирования C и C++;

— общие принципы организации и обработки сложных структур данных (списки, очереди, стеки, деревья) и уметь строить производные структуры данных;

— основы разработки оконных приложений для операционной системы Microsoft Windows 95/98/NT/2000/XP с использованием базового

уровня 32-разрядного интерфейса прикладного программирования приложений (Win32 API);

— основы объектно-ориентированного анализа задач и проектирования приложений.

Краткие сведения из теории

Для выполнения курсовой работы необходимо обратить внимание на следующие теоретические постулаты объектно-ориентированной парадигмы программирования, основанные на методах объектно-ориентированного анализа (ООА) задач и объектно-ориентрованного проектирования (ООП) приложений, предложенных Г. Бучем в работе «Объектно-ориентированный анализ и проектирование».

При объектно-ориентированном анализе разделяют логическую и физическую модели. Логическая модель или логическое представление описывает перечень и смысл ключевых абстракций и механизмов, которые формируют предметную область или определяют архитектуру системы. Физическая модель определяет конкретную программно-аппаратную платформу, на которой реализована или должна будет быть реализована система. Также разделяют статическую и динамическую модель системы, поскольку практически во всех системах происходят события: создаются и уничтожаются объекты, они посылают друг другу сообщения, причем в определенном порядке, внешние события вызывают операции объектов.

При анализе необходимо ответить на следующие вопросы:

— Каково требуемое поведение системы?

— Каковы роли и обязанности объектов по поддержанию этого поведения?

Чтобы выразить решения о поведении системы используются сценариями. В логической модели важнейшим средством для описания сценариев служат диаграммы объектов. При анализе могут быть полезны

диаграммы классов, позволяющие увидеть общие роли и обязанности объектов.

При проектировании необходимо ответить на следующие вопросы относительно архитектуры системы:

— Какие существуют классы и какие есть между ними связи?

— Какие механизмы регулируют взаимодействие классов?

— Где должен быть объявлен каждый класс?

— Как распределить процессы по процессорам и как организовать работу каждого процессора, если требуется обработка нескольких процессов?

Для выражения ответов на эти вопросы, применяются следующие диаграммы:

— диаграммы классов;

— диаграммы объектов;

— диаграммы модулей;

— диаграммы процессов.

Для отражения в объектно-ориентрованном проектировании динамической семантики системы, используются две дополнительные диаграммы:

— диаграммы переходов из одного состояния в другое (также именуемые диаграммами состояний и переходов или кратко диаграммами состояний);

— диаграммами взаимодействия.

Каждый класс может иметь собственную диаграмму переходов, которая показывает, как объект класса переходит из состояния в состояние под воздействием событий. По диаграмме объектов, имея сценарий, можно построить диаграмму взаимодействий, чтобы показать порядок передачи сообщений.

Рассмотрим более подробно диаграммы, которые необходимо выполнить перед переходом собственно к программированию приложения при выполнении данной курсовой работы.

Диаграмма классов

Диаграмма классов показывает классы и их отношения, тем самым представляя логический аспект проекта. Отдельная диаграмма классов представляет определенный ракурс структуры классов. Для полного описания проекта необходимо использовать совокупность диаграмм классов или единую диаграмму для относительно небольших проектов. На стадии анализа диаграммы классов используются, чтобы выделить общие роли и обязанности сущностей, обеспечивающих требуемое поведение системы. На стадии проектирования диаграммы классов используются, чтобы передать структуру классов, формирующих архитектуру системы.

Два главных элемента диаграммы классов — это классы и их основные отношения.

Классы. На рис. 1 показано обозначение для представления класса на диаграмме. Класс обычно представляют аморфным объектом, вроде облака.

Каждый класс должен иметь имя; если имя слишком длинно, его можно сократить или увеличить сам значок на диаграмме. Имя каждого класса должно быть уникально в содержащей его категории.

Рис. 1. Значок класса

Для некоторых языков программирования каждый класс должен иметь имя, уникальное во всей системе. Дл языка C++ имя класса должно быть уникально в рамках содержащего его пространства имен, границы которого

желательно должны совпадать с границей соответствующей категории классов.

На некоторых значках классов полезно перечислять основные атрибуты и операций класса, поскольку для большинства тривиальных классов они бывают очевидны из названия класса и указываются только при предъявлении специальных требований к оформлению диаграммы, например при передаче модели проектирования между исполнителями в качестве конструкторской документации. Атрибуты и операции на диаграмме представляют прообраз полной спецификации класса, в которой объявляются все его элементы. Если на диаграмме необходимо отобразить больше атрибутов класса, можно увеличить значок; если в отображении атрибутов вообще нет необходимости — удаляется разделяющая черта и записывается только имя класса.

Атрибут обозначает часть составного объекта, или агрегата. Атрибуты используются в анализе и проектировании для выражения отдельных свойств класса. На диаграммах классов в нотации Буча используется следующий не зависящий от языка синтаксис, в котором атрибут может обозначаться именем или классом, или и тем и другим, и, возможно, иметь значение по умолчанию:

— **A** только имя атрибута;

— **:C** только класс;

— **A:C** имя и класс;

— **A:C=E** имя, класс и значение по умолчанию.

Имя атрибута должно быть недвусмысленно в контексте класса, то есть имя атрибута должно однозначно характеризовать его назначение в рамках данного класса и не должны требоваться комментарии, предупреждающие о том, что данный атрибут не выполняет какой-либо функции, предполагаемой из его имени. При этом допускается различное функциональное назначение

одноименных атрибутов в различных классах, если они не связаны отношением наследования.

Операции обычно изображаются внутри значка класса только своим именем. Чтобы отличать их от атрибутов, к их именам добавляются скобки. Иногда полезно указать полную сигнатуру операции:

— **N()** только имя операции;

— **RN(Аргументы)** класс возвращаемого значения **(R)**, имя и формальные параметры (если есть).

Имена операций должны пониматься в контексте класса однозначно в соответствии с правилами перегрузки операций выбранного языка реализации.

Общий принцип системы обозначений: синтаксис элементов, таких, как атрибуты и операции, может быть приспособлен к синтаксису выбранного языка программирования. Например, на C++ можно объявить некоторые атрибуты как статические, или некоторые операции как виртуальные или чисто виртуальные; в CLOS моно пометить операцию как метод **:around**. В любом случае используется специфика синтаксиса конкретного языка, чтобы обозначить детали.

Для представления вторичной информации о некой сущности в системе используются «украшения классов», имеющие вид треугольника с буквой внутри, помещаемого внутрь значка класса.

Абстрактный класс — это класс, который не может иметь экземпляров. Так как абстрактные классы очень важны для проектирования хорошей структуры классов, для их обозначения водится украшение в виде значка треугольной формы с буквой A в середине.

Отношения между классами. Классы редко бывают изолированы; напротив, они вступают в отношения друг с другом. Виды отношений: ассоциация, наследование, агрегация (has) и использование. При изображении конкретной связи ей можно сопоставить текстовую пометку,

документирующую имя этой связи или подсказывающую ее роль. Имя связи не обязано быть глобальным, но должно быть уникально в своем контексте.

Значок **ассоциации** соединяет два класса линией (по-возможности, прямой) без дополнительных обозначений на концах и означает наличие семантической связи между ними. Ассоциации часто отмечаются существительными, например **Employment** (место работы), описывающими природу связи. Класс может иметь ассоциацию с самим собой (так называемая рефлексивная ассоциация). Одна пара классов может иметь более одной ассоциативной связи. Возле значка ассоциации вы можете указать ее мощность, используя синтаксис следующих примеров:

— **1** В точности одна связь

— **N** Неограниченное число (0 или больше)

— **0..N** Ноль или больше

— **1..N** Одна или больше

— **0..1** Ноль или одна

— **3..7** Указанный интервал

— **1..3, 7** Указанный интервал или точное число

Обозначение мощности пишется у конца линии ассоциации и означает число связей между каждым экземпляром класса в начале линии с экземплярами класса в ее конце. Если мощность явно не указана, то подразумевается, что она не определена.

Обозначения оставшихся трех типов связи уточняют значок ассоциации (просто линию) дополнительными пометками в местах соединения изображения связи со значками классов. Это удобно, так как в процессе разработки проекта связи имеют тенденцию уточняться. Сначала устанавливается семантическая связь между двумя классами, а потом, после принятия тактических решений об истинных их отношениях, уточняем эту связь как наследование, агрегацию или использование. Возможна ситуация,

когда прямая ассоциация между двумя классами реализуется не непосредственно, а через другие классы, с которыми установлены отношения агрегации, наследования или использования.

Значок **наследования**, представляющего отношение "общее/частное", выглядит как значок ассоциации со стрелкой, которая указывает от подкласса к суперклассу. В соответствии с правилами выбранного языка реализации, подкласс наследует структуру и поведение своего суперкласса. Класс может иметь один (одиночное наследование), или несколько (множественное наследование) суперклассов. Конфликты имен между суперклассами разрешаются в соответствии с правилами выбранного языка. Как правило, циклы в наследовании запрещаются. К наследованию значок мощности не приписывается.

Значок **агрегации** обозначает отношение "целое/часть" (связь "has") и получается из значка ассоциации добавлением закрашенного кружка на конце, обозначающем агрегат. Экземпляры класса на другом конце стрелки будут в каком-то смысле частями экземпляров класса-агрегата. Разрешается рефлексивная и циклическая агрегация. Агрегация не требует обязательного физического включения части в целое. Способ реализации агрегации отмечается на диаграмме украшением на конце линии, обозначающей агрегацию; отсутствие этого украшения означает, что решение о способе реализации не определено. В гибридных языках различают два типа агрегации:

— по значению: целое физически содержит часть

— по ссылке: целое физически содержит указатель или ссылку на часть.

В чисто объектно-ориентированных языках, в особенности в Smalltalk или Python, физическое реализация агрегации бывает только по ссылке.

Чтобы отличить физическое присутствие объекта от ссылки на него, используется закрашенный квадратик для обозначения агрегации по значению и пустой квадратик — для агрегации по ссылке.

Знак **использования** обозначает отношение "клиент/сервер" и изображается как ассоциация с пустым кружком на конце, соответствующем клиенту. Эта связь означает, что клиент нуждается в услугах сервера, то есть операции класса-клиента вызывают операции класса-сервера или имеют сигнатуру, в которой возвращаемое значение или аргументы принадлежат классу сервера.

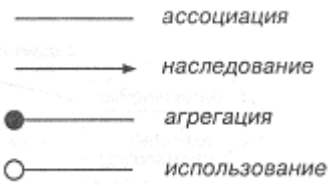

Рис. 2. Обозначения отношений между классами

Диаграммы состояний и переходов

Диаграмма состоянии и переходов показывает: пространство состояний данного класса; события, которые влекут переход из одного состояния в другое; действия, которые происходят при изменении состояния. Отдельная диаграмма состояний и переходов представляет определенный ракурс динамической модели отдельного класса или целой системы. Диаграммы состояний и переходов имеет смысл строить только для классов, поведение которых (управляемое событиями) существенно для рассматриваемой системы. Возможно представить диаграмму состояний и переходов для управляемого событиями поведения системы в целом. Эти диаграммы используются в ходе анализа, чтобы показать динамику поведения системы, а в ходе проектирования — для выражения поведения отдельных классов или их взаимодействия.

Два основных элемента диаграммы состояний и переходов — это, естественно, состояния и переходы между ними.

Состояние представляет собой итоговый результат поведения системы. Например, только что включенный в сеть телефон находится в начальном состоянии: его предыдущее поведение несущественно, при этом он готов к тому, чтобы позвонить или принять звонок. Если кто-нибудь поднимет трубку, телефон перейдет в состояние готовности к набору номера; в этом состоянии не ожидается, что телефон зазвонит, но поднявший трубку приготовился к беседе с одним или несколькими абонентами. Если кто-либо наберет этого номер, а телефон находится в начальном состоянии (трубка положена), то когда кто-нибудь поднимет трубку, телефон перейдет в состояние с установленным соединением, и поднявший трубку сможет поговорить со звонившим.

В любой момент времени состояние объекта определяет набор свойств (обычно статический) объекта и текущие (обычно динамические) значения этих свойств. Под "свойствами" подразумевается совокупность всех связей и атрибутов объекта. Можно обобщить понятие состояния так, чтобы оно было применимо и к объекту, и к классу, так как все объекты одного класса "живут" в одном пространстве состояний. Это пространство может представлять собой неопределенное, хотя конечное множество возможных (но не всегда ожидаемых или желаемых) состояний. На рис. 3 показано обозначение, которое используется для отдельного состояния.

Рис. 3. Значок состояния

Каждое состояние должно иметь имя; если оно оказывается слишком длинным, то его можно сократить или увеличить значок состояния. Каждое имя состояния должно быть уникально в своем классе. Состояния, ассоциированные со всей системой, глобальны, то есть видимы отовсюду, а

область видимости вложенных состояний ограничена соответствующей подсистемой. Все одноименные значки состояний на одной диаграмме обозначают одно и то же состояние.

На значках некоторых состояний полезно указать ассоциированные с ними действия. Как показано на рис. 3, действия обозначаются так же, как атрибуты и операции в значке класса. Можно увеличить значок, чтобы увидеть весь список действий, или, если нет необходимости указывать действия, можно удалить разделяющую линию и оставить только имя.

Переходы. Событием называется любое происшествие, которое может быть причиной изменения состояния системы. Изменение состояний называется переходом. На диаграмме переходов и состояний он изображается значком, показанным на рис. 4. Каждый переход соединяет два состояния.

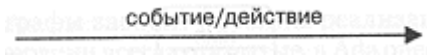

Рис. 4. Значок перехода из состояния в состояние

Состояние может иметь переход само в себя; обычно есть несколько различных переходов в одно и тоже состояние, но все переходы должны быть уникальны в том смысле, что ни при каких обстоятельствах не может произойти одновременно два перехода из одного состояния.

Действием называется операция, которая, с практической точки зрения, требует нулевого времени на выполнение. Например, включение сигнала тревоги — действие. Обычно действие означает вызов метода, порождение другого события, запуск или остановку процесса. Деятельностью называется операция, требующая некоторого времени на свое выполнение. Например, нагрев воздуха в теплице — деятельность, запускаемая включением нагревателя, который может оставаться включенным неопределенное время, до тех пор, пока не будет выключен явной командой.

При анализе можно давать предварительные названия событиям и действиям, в общих чертах отражая понимание предметной области. Однако,

отображая эти понятия на классы, необходимо предложить конкретную стратегию реализации.

Событие может быть представлено символическим именем (или именованным объектом), классом или именем некоторой операции. Например, в рамках конкретной системы можно придерживаться той стратегии, что все события являются символическими именами и каждый класс с поведением, управляемым событиями, имеет операцию, которая распознает эти имена и выполняет соответствующие действия. Для большей общности можно считать события объектами и определить иерархию классов, которые представляют собой абстракции этих событий. Продолжая рассуждать аналогично, можно сделать так, чтобы переходы из состояния в состояние были полиморфны относительно классов событий. Наконец, можно определить событие просто как операцию класса-получателя события. Это похоже на подход, который трактует события как имена, но в отличие от него здесь не требуется явного диспетчера событий.

Для нашего метода несущественно, какая из этих стратегий выбрана для разработки, если она последовательно проводится во всей системе. Обычно в замечаниях указывается, какая стратегия использована для данного конкретного автомата.

Действие можно записывать, используя синтаксис, показанный в следующих примерах:

— **heater.startUp()** действие
— **DeviceFailure** произошло событие
— **start Heating** начать некоторую деятельность
— **stop Heating** прекратить деятельность.

Имена операций или событий должны быть уникальны в области видимости диаграммы; там, где необходимо, они могут быть квалифицированы соответствующими именами классов или объектов. В случае начала или прекращения некоторой деятельности, она может быть представлена операцией (такой, как **Actuator::shutDown ()**) или

символическим именем (для событий). Когда деятельность соответствует некоторой функции системы, такой, как **harvest crop** (сбор урожая), обычно пользуются символическими именами.

На каждой диаграмме состояний и переходов должно присутствовать ровно одно стартовое состояние; оно обозначается немаркированным переходом в него из специального значка, изображаемого в виде закрашенного кружка. Пример диаграммы состояний и переходов изображен на рис. 5.

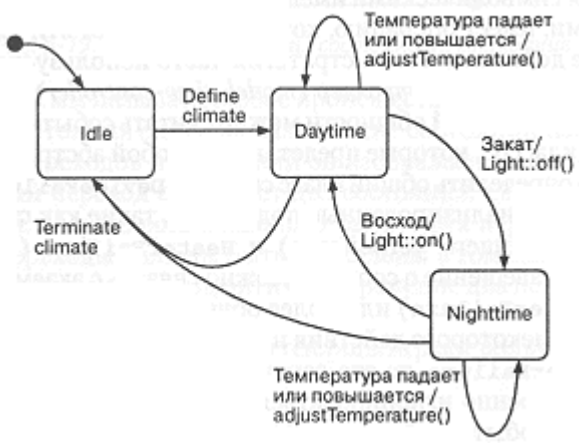

Рис. 5. Диаграмма состояний и переходов для контроллера тепличной среды (класс **EnvironmentalController**)

Иногда бывает нужно указать также конечное состояние (обычно автомат, ассоциированный с классом или системой в целом, никогда не достигает конечного состояния; этот автомат просто перестает существовать после того, как содержащий его объект уничтожается). Конечное состояние обозначается немаркированным переходом от него к специальному значку, изображаемому как кружок с закрашенной серединой.

Действия, ассоциированные с состояниями и условные переходы. Как показано на рис. 3, с состояниями могут быть ассоциированы действия. В частности, можно назначить выполнение некоторого действия на входе или выходе из состояния, при этом используется синтаксис следующих примеров:

— **entry start Alarm** запуск процедуры при входе в состояние

— **exit shutDown()** вызов операции при выходе из состояния.

Как и для переходов, можно назначить любое действие после ключевых слов **entry** и **exit**(вход и выход).

Деятельность можно ассоциировать с состоянием, используя синтаксис следующего примера

— **do Cooling** в данном состоянии заниматься этой деятельностью.

Этот синтаксис служит сокращенной записью явных указаний: "Начать деятельность при входе в состояние и окончить при выходе из него".

На рис. 6 приведен пример использования этих обозначений. При входе в состояние **Heating** (нагревание) вызывается операция **Heater::startUp()**, а при выходе - операция **Heater::shutDown()**, то есть происходит запуск и остановка нагревания. При входе и выходе из состояния **Failure** (сбой), соответственно вызывается и прекращается сигнал тревоги (**Alarm**).

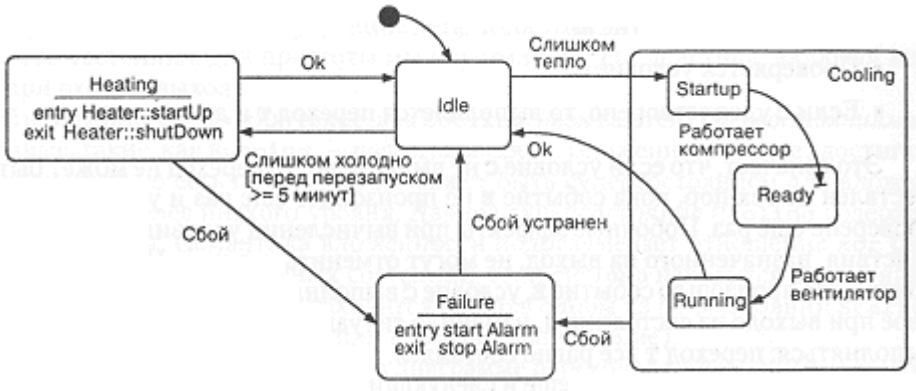

Рис. 6. Действия, условные переходы и вложенные состояния

Рассмотрим также переход из состояния **Idle** в состояние **Heating**. Он совершается, если температура понизилась, но только в случае, если прошло больше пяти минут после того, как последний раз был выключен нагреватель. Это пример условного (или защищенного) перехода; условие представляется логическим выражением в скобках.

Вообще, каждый переход может быть ассоциирован либо с событием, либо с событием и условием. Допускаются и "переходы без события". В этом случае переход совершается сразу после завершения действия, связанного с состоянием, причем выполняется и действие, связанное с выходом из этого состояния. Если переход условный, он состоится только в случае, если условие выполнено.

Имеет значение порядок выполнения условного перехода. Пусть имеется состояние **S**, из которого при событии **E** совершается переход **T** с условием **C** и действием **A**. Переход **T** осуществляется в такой последовательности:

— Происходит событие **E**.

— Проверяется условие **C**.

— Если **C** удовлетворено, то выполняется переход **T** и действие **A**.

Это означает, что если условие **C** не выполнено, то переход не может быть осуществлен до тех пор, пока событие **E** не произойдет еще раз и условие **C** не будет проверено еще раз. Побочные эффекты при вычислении условия или выполнении действия, назначенного на выход, не могут отменить переход. Например, предположим, что произошло событие **E**, условие **C** выполнилось, но действие **A**, выполняемое при выходе из состояния **S**, изменило ситуацию так, что условие **C** перестало выполняться: переход **T** все равно состоялся.

Также может использоваться следующий синтаксис:

— **in Cooling** выражение для текущего состояния.

Здесь используется имя состояния (которое может быть квалифицированным). Выражение истинно тогда и только тогда, когда система находится в указанном состоянии. Такие условия особенно полезны, когда некоторому внешнему состоянию нужно запустить переход по условию, связанному с некоторым вложенным состоянием.

Можно использовать в условии и выражение, налагающее ограничения по времени:

— **timeout (Heating, 30)** выражение ограничения по времени.

Это условие выполняется, если система более 30 секунд находилась в состоянии **Heating** и остается в нем в момент проверки. Этот тип условия употребляется в системах реального времени для "переходов без события", так как защищает систему от зависания на долгое время в одном состоянии. Это выражение можно использовать для указания нижней границы времени нахождения в данном состоянии. Если приложить временное ограничение к каждому переходу с событием, выводящим из данного состояния, это будет равнозначно требованию, что система находится в каждом состоянии как минимум время, указанное в ограничении.

Ситуацию, когда происходит некое событие, а перейти в другое состояние нельзя либо потому, что не существует перехода для данного события, либо не выполняется условие перехода по умолчанию надо считать ошибкой: игнорирование событий обычно является признаком неполного анализа задачи. Вообще, для каждого состояния нужно документировать события, которые оно намеренно игнорирует.

Вложенные состояния. Возможность вложения состояний друг в друга придает глубину диаграммам переходов; эта ключевая особенность предотвращает комбинаторный взрыв в структуре состояний и переходов, который часто случается в сложных системах.

На рис. 6 показаны внутренние детали состояния **Cooling**, то есть вложенные в него состояния; для простоты были опущены все его действия, включая действия при входе и выходе.

Объемлющие состояния, такие, как **Cooling**, называются суперсостояниями, а вложенные, такие, как **Running**, — подсостояниями. Вложенность может достигать любой глубины, то есть подсостояние может быть суперсостоянием для вложенных состояний более низкого уровня. Данное суперсостояние **Cooling** содержит три подсостояния. Семантика

вложенности подразумевает отношение **xor** (исключающее или) для вложенных состояний: если система находится в состоянии **Cooling** (охлаждение), то она находится ровно в одном из подсостояний **Startup** (начальное), **Ready** (готовность) или **Running** (выполнение).

Чтобы проще ориентироваться в диаграмме переходов с вложенными состояниями можно увеличить или уменьшить ее масштаб относительно выбранного состояния. При уменьшении вложенные состояния исчезают, а при увеличении проявляются. Переходы в скрытые на диаграмме подсостояния и выходы из них показываются стрелкой с черточкой, как переход в состояние **Ready** на рис. 6.

Переходам между состояниями разрешено начинаться и кончаться на любом уровне. Рассмотрим различные формы переходов:

— Переход между одноуровневыми состояниями (такой, как из **Failure** В **Idle** или из **Ready** в **Running**) — простейшая форма перехода; его семантика описана в предыдущем разделе.

— Можно совершить переход непосредственно в подсостояние (как из **Idle** в **Startup**), или непосредственно из подсостояния (как из **Running** в **Idle**), или одновременно и то, и другое.

— Указание перехода из суперсостояния (как из **Cooling** в **Failure** через событие **Failure**) означает, что он осуществляется из каждого подсостояния этого суперсостояния. Такой переход пронизывает все уровни до переопределения. Это упрощает диаграмму за счет удаления банальных переходов, общих для всех подсостояний.

Указание перехода в состояние с вложенными подсостояниями (например, предыдущий переход в состояние **Failure**) подразумевает переход к его начальному подсостоянию (по умолчанию).

Диаграммы объектов

Диаграмма объектов показывает существующие объекты и их связи в логическом проекте системы. Иначе говоря, диаграмма объектов

представляет собой мгновенный снимок потока событий в некоторой конфигурации объектов. Таким образом, диаграммы объектов являются своего рода прототипами: каждая представляет взаимодействие или структурные связи, которые могут возникнуть у данного множества экземпляров классов, безотносительно к тому, какие конкретно экземпляры участвуют в этом взаимодействии. В таком смысле, отдельная диаграмма объектов есть ракурс структуры объектов системы. При анализе диаграммы объектов используются для показа семантики основных и второстепенных сценариев, которые отслеживают поведение системы. При проектировании диаграммы объектов используются для иллюстрации семантики механизмов в логическом проектировании системы.

Объекты. На рис. 7 показан значок, который изображает объект на диаграмме объектов. Как и в диаграммах классов, можно провести горизонтальную линию, разделяющую текст внутри значка объекта на две части: имя объекта и его атрибуты.

Рис. 7. Значок объекта на диаграмме объектов

Имя объекта следует синтаксису для атрибутов, и может быть либо записано в одной из трех следующих форм:

- **A** только имя объекта
- **:C** только класс объектов
- **A:C** имя объекта и класса

либо использовать синтаксис выбранного языка реализации. Если текст не умещается внутри значка, то следует или увеличить значок, или сократить текст. Если несколько значков объектов на одной диаграмме используют

одно и то же неквалифицированное имя (то есть имя без указания класса), то они означают один и тот же объект. В противном случае каждый значок означает отдельный объект. Если на разных диаграммах есть объекты с одинаковыми неквалифицированными именами, то это разные объекты, если только их имена не квалифицированы явно.

Смысл неквалифицированных имен зависит от контекста диаграммы объектов. Более точно: диаграммы объектов, определенные на самом верхнем уровне системы, имеют глобальную область видимости; другие диаграммы объектов могут быть определены для категорий классов, отдельных классов или отдельных методов, а, значит, иметь соответствующие области видимости. Квалифицированное имя может быть использовано при необходимости явной ссылки на глобальные объекты, переменные классов (статические элементы в C++), параметры методов, атрибуты или локально определенные объекты в той же области видимости.

Если не указать класс объекта — ни явно, использовав ранее упомянутый синтаксис, ни косвенно, через спецификацию объекта, — то класс рассматривается как анонимный, при этом нельзя провести семантическую проверку ни операций, совершаемых над объектом, ни его связей с другими объектами на диаграмме. Если же указать только класс, то анонимным считается объект. Каждый значок без имени объекта обозначает отдельный анонимный объект.

В любом случае, имя класса объекта должно быть именем настоящего класса (или любого из его суперклассов) в области видимости диаграммы, использованного для инстанцирования объекта (создания конкретного объекта), даже если этот класс — абстрактный. Эти правила позволяют написать сценарий, не зная точно, к каким подклассам принадлежат объекты. На значках объектов бывает полезно указать несколько их атрибутов. "Несколько" — так как значок объекта представляет только один какой-то ракурс его структуры. Синтаксис атрибутов совпадает с описанным ранее

синтаксисом атрибутов класса и позволяет указать выражение, используемое по умолчанию. Имена атрибутов объектов должны соответствовать атрибутам, определенным в классе объекта, или в любом из его суперклассов. Синтаксис имен атрибутов может быть приспособлен к синтаксису языка реализации.

Диаграмма объектов может также включать значки, обозначающие утилиты классов и метаклассы: эти понятия подобны объектам, так как они могут действовать как объекты, и с ними можно оперировать как с объектами.

Отношения между объектами. Объекты взаимодействуют с другими объектами через связи, обозначение которых показано на рис. 8. Подобно тому, как объект является экземпляром класса, связь является экземпляром ассоциации.

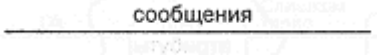

Рис. 8. Значок связи между объектами на диаграмме объектов

Связь между двумя объектами (включая утилиты классов и метаклассы) может существовать тогда и только тогда, когда существует ассоциация между соответствующими классами. Ассоциация между классами может проявляться различными способами, например, как простая ассоциация, отношение наследования или отношение включения. Следовательно, существование ассоциаций между двумя классами означает существование коммуникации (то есть канала связи) между их экземплярами, по которой объекты могут посылать друг другу сообщения. Все классы неявно имеют ассоциации сами с собой и, следовательно, объект может послать сообщение сам себе.

Пусть имеются объекты **A** и **B** и связь **L** между ними. Тогда **A** может вызвать любую операцию, имеющуюся в классе **B** и доступную **A**. То же

верно для операций над **A**, вызываемых **B**. Объект, вызывающий операцию, называется объект-клиент, а объект, который предоставляет операцию, - объект-сервер. Обычно отправитель сообщения знает получателя, но обратное необязательно.

В установившемся состоянии структуры классов и объектов системы должны быть согласованы. Если на диаграмме показывается операция **M** на классе **B**, вызванная по связи **L**, то **M** должна быть объявлена в спецификации **B**, или в спецификациях его суперклассов.

Как показано на рис. 8, рядом с соответствующей связью на диаграмме можно записать набор сообщений. Каждое сообщение состоит из следующих трех элементов:

— **D** символ синхронизации, обозначающий направление вызова

— **M** вызов операции или извещение о событии

— **S** необязательный порядковый номер.

Направление сообщения показывается стрелкой, указывающей на объект-сервер. Этот символ означает простейшую форму передачи сообщений, семантика которой гарантирована только в присутствии единственного потока управления функционированием системы. Существуют более развитые формы синхронизации, которые применимы в случае нескольких потоков.

Вызов операции — наиболее общая форма сообщения. Она подчиняется ранее описанному синтаксису операций, но, в отличие от него, здесь могут быть приведены фактические параметры, подходящие к сигнатуре операции:

— **N()** только имя операции

— **RN(arguments)** возвращаемое значение, имя и фактические параметры операции.

Сопоставление фактических параметров с формальными осуществляется в порядке следования. Если возвращаемый операцией объект

или фактические параметры используют неквалифицированные имена, совпадающие с другими неквалифицированными именами на диаграмме, то подразумевается, что они именуют одинаковые объекты, а следовательно, их классы должны подходить к сигнатуре операции. Таким образом, возможно представление взаимодействий, в ходе которых объекты передаются в качестве параметров или возвращаются, как результат операции.

Сообщение может извещать о событии. Оно подчиняется определенному ранее синтаксису событий, и, следовательно, может представлять символьное имя, объект или имя некоторой операции. Во всяком случае, имя события должно быть определено на соответствующей классу объекта-сервера диаграмме состояний и переходов. Если извещение о событии является операцией, то оно может включать фактические параметры.

Если порядковый номер явно не указан, то сообщение может быть послано независимо от других сообщений, указанных на данной диаграмме объектов. Чтобы указать явный порядок событий, их можно пронумеровать. Нумерация начинается с единицы и добавляется как необязательный префикс к вызову операции или извещению о событии. Порядковый номер показывает относительный порядок посылки сообщений. Сообщения с одинаковыми номерами не упорядочены друг относительно друга; сообщение с меньшим порядковым номером посылается до сообщения с большим номером. Повторение порядковых номеров или их отсутствие говорит о частичной упорядоченности сообщений.

Таким образом, диаграмма объектов позволяет представить поведение системы на уровне взаимодействия конкретных объектов тех или иных классов, что существенно упрощает в дальнейшем переход от логической модели анализа и проектирования к физической модели реализации.

Диаграмма модулей

Диаграмма модулей показывает распределение классов и объектов по модулям в физическом проектировании системы. Каждая отдельная диаграмма модулей представляет некоторый ракурс структуры модулей системы. При разработке диаграмму модулей используют, чтобы показать физическое деление архитектуры по слоям и разделам. Некоторые языки, особенно Smalltalk, не имеют ничего подобного физической архитектуре, сформированной модулями; в таких случаях диаграммы модулей не употребляют. Основными элементами диаграммы модулей являются модули и их зависимости.

Модули. На рис. 9 сведены обозначения различных типов модулей. Первые три значка — это файлы, различающиеся своими функциями. Значок главной программы обозначает файл, содержащий корневую программу (содержащий точку начала функционирования системы). В C++, например, это соответствовало бы некоторому файлу с расширением **.cpp** содержащему привилегированную функцию-неэлемент, называемую **main**. Обычно существует ровно один такой модуль на программу. Значок описания и значок тела обозначают файлы, которые содержат, соответственно, описания и реализации. Каждый модуль содержит либо описание, либо определение классов и объектов, а также другие конструкции языка. В C++, например, модуль описаний соответствует заголовочному файлу с расширением **.h**, а модуль тела — файлу с текстом программы с расширением **.cpp**. Каждый модуль должен иметь имя; обычно это имя соответствующего физического файла в каталоге проекта. Как правило, такие имена пишутся без суффиксов, которые опознаются по типу значка. Если имя чересчур длинно, либо оно сокращается, либо расширяется значок. Каждое полное имя файла должно быть уникально в содержащей его подсистеме. В соответствии с правилами конкретной среды разработки, можно наложить ограничения на имена, такие, как условие на префиксы или требование уникальности в системе.

Подсистемы. Подсистемы служат частями физической модели системы. Подсистема — это агрегат, содержащий другие модули и другие

подсистемы. Каждый модуль в системе должен жить в одной подсистеме или находиться на самом верхнем уровне.

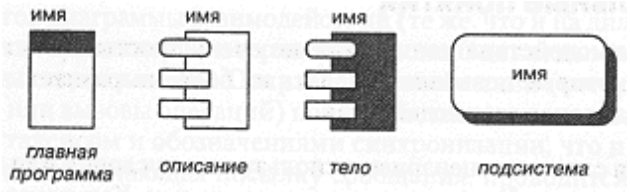

Рис. 9. Значки модулей и подсистем.

На рис. 9 также показано обозначение подсистемы. Как и модуль, подсистема должна быть именованной. Имена подсистем подчиняются тем же правилам, что и имена модулей, хотя полное имя подсистемы обычно не содержит суффиксов.

Некоторые модули, содержащиеся в подсистеме, могут быть общедоступны, то есть экспортированы из системы и видимы снаружи. Другие модули могут быть частью реализации подсистемы и не предназначаться для использования внешними модулями. По соглашению, каждый модуль подсистемы считается общедоступным, если явно не указано обратное. Ограничение доступа к модулям реализации достигается использованием тех же обозначений, что и для ограничения доступа в категории классов.

Зависимости. Единственная связь, которая может существовать между двумя модулями, — компиляционная зависимость — представляется стрелкой, выходящей из зависимого модуля. В C++, например, такая зависимость указывают директивой **#include**. Аналогично, в Ada компиляционная зависимость указывается фразой **with**. В множестве компиляционных зависимостей не могут встречаться циклы. Чтобы определить частичную упорядоченность компиляций, достаточно выполнить частичное упорядочение структуры модулей системы. Подсистема также может зависеть от других подсистем и модулей; модуль может также зависеть от подсистемы. Для единообразия используется обозначение зависимости подсистем, аналогичное обозначению зависимости модулей.

Система имеет один высший уровень, состоящий из подсистем и модулей высшего уровня абстракции. По его диаграмме разработчик получает представление об общей физической архитектуре системы.

Пример. На рис. 10 показан пример обозначений модулей, в архитектуре системы тепличного гидропонного хозяйства.

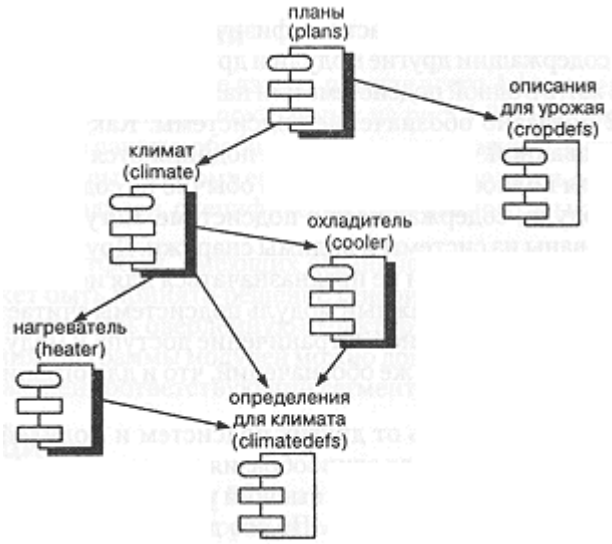

Рис. 10. Диаграмма модулей гидропонной системы.

Два модуля, **climatedefs** и **cropdefs**, являются только описаниями и служат для предоставления общих типов и констант. Остальные четыре включают в себя и тела, и описания: это типичный стиль построения диаграмм модулей, так как описания и тела очень тесно связаны. На рисунке эти две части совмещены, и зависимость тела от описания получилась скрытой, хотя реально она существует. Также оказалось скрытым имя тела, но, по распространенному для языка C++ соглашению, имена тела и описания различаются лишь суффиксами (**.cpp** и **.h**). Зависимости на этой диаграмме предполагают частичное упорядочение компиляции. Например, тело модуля **climate** зависит от описания **heater**, которое, в свою очередь, зависит от описания **climatedefs**.

Рассмотренные четыре вида диаграмм позволяют описать наиболее важные аспекты логической и физической моделей анализируемой и проектируемой системы при использовании объектно-ориентированного

подход и инструментальных средств, реализующих объектно-ориентированную парадигму программирования.

Однако, несмотря на высокую степень общности и универсальности объектно-ориентированной парадигмы, в определенных случаях необходимо непосредственное применение инструментальных средств, поддерживающих только парадигму структурного программирования. В настоящее время, наиболее распространенной ситуацией, требующей применения структурного подхода является разработка программ, непосредственно использующих интерфейсы прикладного программирования операционных систем, стандартом де-факто для описания которых является язык программирования C. Одним из примеров такого интерфейса является Win32 API, предназначенный для создания прикладных программ для 32-разрядных версий операционных систем Microsoft Windows и являющийся общим для всех операционных систем от Windows 95 и Windows NT до Windows XP и Windows 2003 Server.

Основной логической особенностью использования Win32 API является то, что с одной стороны, операционные системы Microsoft Windows относятся к ОС, управляемым событиями и их многие внутренние сущности выражаются в виде объектов, а с другой стороны, он реализован на процедурном языке C.

Поэтому, программирование с использованием Win32 API требует хорошего понимания особенностей выполнения функций, вызываемых по указателям (например, основная функция обработки сообщений окна), зависимости использования структур от параметров компиляции, а самое главное, учитывая многозадачность и многопоточность ОС Windows, понимания механизмов разграничения доступа к различным сущностям программы извне и внутри неё.

Поскольку Win32 API содержит более тысячи функций, констант, структур и других элементов, из которых для создания минимальной

программы необходимо около двух десятков, а программы выполняющей какие-либо полезные действия — около сотни, то разработка программ с его помощью требует постоянного обращения к справочной литературе и справочной системе инструментальных средств разработки, в особенности к Microsoft Developer Network.

Последовательность этапов выполнения курсовой работы

1. Получение индивидуального задания на курсовую работу — не позднее 2 недели семестра;

2. Неформальное описание предлагаемого решения задачи, уточнение задания, разрешение вопросов по источникам дополнительной информации, получение от преподавателя подтверждения правильности понимания задания — не позднее 3 недели;

3. Создание каркаса пользовательского интерфейса приложения как работающей программы — не позднее 5 недели;

4. Объектно-ориентированный анализ задачи, проектирование основных элементов реализации. При этом рекомендуется перед написанием текстов программы даже для очевидных элементов системы создать соответствующие диаграммы, которые в дальнейшем будут уточняться в процессе реализации. Рекомендуется предъявление работоспособных элементов основной функциональности приложения (возможно, с консольным «отладочным» интерфейсом) преподавателю не позднее 7 недели семестра;

5. Сведение функциональной и интерфейсной частей программы рекомендуется завершить предъявлением программного обеспечения преподавателю не позднее 11 недели семестра, после чего приступить к выполнению отчета;

6. Предъявление предварительной версии отчета по курсовой работе рекомендуется выполнить не позднее 13 недели, чтобы оставалось время на устранение замечаний;

7. Устранение замечаний и повторное предъявление в работе программного обеспечения и отчета по курсовой работе рекомендуется выполнить не позднее 15 недели;

8. Выполнение курсовой работы завершается её защитой перед преподавателем, что необходимо выполнить не позднее 17 недели, так как защищенная курсовая работа является необходимой для получения допуска к экзамену.

Перед предъявлением оформленной пояснительной записки к курсовой работе, программа, разрабатываемая согласно варианту задания на курсовую работу, должна быть предъявлена в рабочем состоянии после трансляции и компоновки в аудитории по расписанию занятий.

Предъявление ПО курсовой работы на собственных технических средствах обучающихся **НЕ ДОПУСКАЕТСЯ!**

Требования к оформлению пояснительной записки

Пояснительная записка выполняется в соответствии с требованиями «Положения о курсовых проектах и курсовых работах на факультете информационных и управляющих систем» и ГОСТ 7.32-2001 Отчет о научно-исследовательской работе: структура и правила оформления. Титульный лист должен быть выполнен по образцу, указанному в приложении к «Положению…»

На месте приведенного в «Положении…» раздела «Содержательная часть» указываются:

1. Краткие сведения из теории, а именно принцип работы ОС, управляемой сообщениями; соотношение языков Си и Си++; описание основных функций, обеспечивающих создание окон и обмен сообщениями в

ОС Windows, а также основные способы реализации принципов объектно-ориентированного программирования.

2. Описание программы, содержательно соответствующее ГОСТ 19.402 Описание программы.

В описании программы, в соответствии со стандартом, указываются:

1. вводная часть – указание на то, что данная программа разработана в рамках курсовой работы по дисциплине «Программирование на языках высокого уровня»;
2. функциональное назначение;
3. **Условия применения** указываются условия, необходимые для выполнения программы (требования к необходимым для данной программы техническим средствам, и другим программам, общие характеристики входной и выходной информации, а также требования и условия организационного, технического и технологического характера и т.п. сюда же входит инструкция пользователя);
4. На месте раздела **«описание логики»** приводятся следующие сведения:

    4.1. описание структуры программы и ее основных частей (например: *В состав программы входит следующее…);*

    4.2. описание функций составных частей и связей между ними (Например*: Программа состоит из шести модулей: интерфейсный модуль; модуль определения ...; модуль расчета ...; модуль ...и т.п.. Интерфейсный модуль построен на двух типах диалогов: диалог "вопрос - ответ" и диалог типа "меню". Интерфейсный модуль управляет ...Модуль определения ... Он является*);

    4.3. сведения о языке программирования (не нужно здесь ДОСЛОВНО цитировать известные книги по С++! Должен быть изложен необходимый минимум сведений — что за язык, кем разработан, согласно какой версии стандарта языка выполнен тот или иной модуль программного обеспечения);

4.4. описание входных и выходных данных для каждой из составных частей;

4.5. описание логики составных частей, в обязательном порядке включающее в себя:

— Диаграмму классов — в отчет включаются диаграммы классов этапа проектирования, причем со всеми классами, написанными студентом, на диаграммах в отчете указываются все атрибуты и операции со спецификаторами доступа **public** и **protected**;

— Диаграмму состояний и переходов;

— Диаграмму модулей;

— Блок-схемы алгоритмов наиболее нетривиальных методов (если есть) или согласно дополнительного указания.

Внимание! НЕЛЬЗЯ называть этот раздел «Описание логики»! Наиболее корректное название — «Организация программы».

Приложения к пояснительной записке по курсовой работе содержат:

1. Приложение 1 — Текст программы по ГОСТ 19.401 (полные тексты всех модулей, необходимых для компиляции и выполнения программы, включая дополнительно разработанные исполнителем библиотеки-обертки и т.д.)
2. Приложение 2 — Руководство оператора по ГОСТ 19.505

Приложения 1 и 2 оформляются согласно ГОСТ 19.106, даже если его требования противоречат ГОСТ 7.32. При этом рамка документа не используется и основная надпись не выполняется, лист утверждения и лист регистрации изменений – отсутствуют за ненадобностью; аннотация должна давать представление о содержании всего следующего за ней документа, выраженное в нескольких предложениях (не более трех-пяти).

Темы курсовых работ

Каждая тема может уточняться при выдаче задания или в течении первого этапа выполнения работы. Для каждой темы указана относительная сложность того или иного подварианта темы. Сложность 0 — для выполнения работы достаточно освоения общих принципов программирования приложений для целевой ОС и соответствующих языков программирования, а также строгого следования стандартам. Задания со сложность 4 и 5 — требуют от студента большого объема самостоятельной работы, возможно включая обработку англоязычных технических описаний, а также развития навыков творческой исследовательской и конструкторской деятельности.

— Игра «жизнь». С «бесконечным»/ограниченным полем, масштабом, цветовым/числовым кодированием «возраста» клеток. Сохранение/восстановление состояния. Возможно расширение сервисных функций. «Снимок экрана» в один из стандартных графических форматов (bmp/pcx/tiff). Сложность от 1 до 3 в зависимости от функций. Для Windows – ScreenSaver на её основе (в дополнение!). Сложность 3

— Утилита построения частотного словаря. С просмотрщиком словарь/текст (подсветка вхождений искомого слова, переход между вхождениями вперед/назад). Построение словаря на основе сбалансированного дерева поиска. (Сложность 1)

— Утилита преобразования исходных текстов на языке С (C++/Dephi/… - какой-нибудь) в текст формате rtf. При этом производится выделение синтаксических конструкций (по фиксированным правилам/правила из текстового файла настройки/правила задаются при интерактивной настройки программы) Сложность 1 или 2 в зависимости от функций по настройке.

— Программа просмотра графических файлов (в формате bmp/pcx/tiff/нескольких форматах). Сложность: только bmp — 0, один другой растровый формат — 1, два растровых формата — 2, преобразование

формата — 3, только распаковка JPEG — 3, распаковка JPEG, просмотр и сохранение в другой формат — 4.

— Программа хранения данных о студентах: ФИО, №зачетки, 5 оценок. Ввод/изменение данных, просмотр табличный/карточками. Сортировку списка по убыванию/возрастанию среднего балла реализовать методом нерекурсивной быстрой сортировки, сортировку по алфавиту реализовать рекрсивным вариантом быстрой сортировки. Сложность 1 или 2 при хорошем интерфейсе И хорошем проектировании содержания (диаграммы классов и т.д.).

— Программа «Записная книжка»: ключевое поле (1 обязательное, до 4 дополнительных) и произвольное текстовое поле (максимальной длинны, допустим, до 4Кб на одну запись). Последовательный доступ к «страницам» книжки, прямой доступ по списку «заглавий» (таблица значений ключевых полей), сортировка по любому сочетанию ключевых полей в указанной последовательности, поиск по ключевым полям с логическими операциями над ними. Реализация нестрогого соответствие значений ключевых полей образцу — сложность 5. Иначе — сложность 1 или 2 при хорошем интерфейсе И хорошем проектировании содержания (диаграммы классов и т.д.).

— «Анализ экспериментальных данных»: построение графика функции, заданной таблично с наложением на неё графиков стандартных функций с указанными параметрами. Возможно одновременное наложение нескольких графиков, удаление функций, изменение параметров графика. Сложность 3. При реализации визуального изменения параметров графика (как в векторном редакторе: выделяем график, к нему строится прямоугольник изменения размера, тянем его, а в результате происходит пересчет параметров графика) — сложность 5.

— Программа символьной математики: раскрытие скобок в арифметическом выражении, приведение подобных членов, определение

тождественности двух арифметических выражений. Сложность 3 или 4 в зависимости от функциональности, интерфейса И качества проектирования .

— Векторный «графический редактор» — сложность 3. С поддержкой WMF — сложность 4. С поддержкой SVG — сложность 5.

— Разбор и отображении графического формата SVG (разбор XML + формирование отнюдь не примитивных примитивов) – документация на АНГЛИЙСКОМ! Сложность 4.

— Разбор и отображение графического формата WMF – документация на АНГЛИЙСКОМ! Сложность 4.

Контрольные вопросы

1. Назовите основные принципы программирования для ОС, управляемой событиями.
2. Приведите типовую структуру основной программы (WinMain) приложения для ОС Windows XP
3. Дайте понятие handle в ОС Windows XP. Дайте понятия окно и класс окна, укажите взаимоотношение между ними.
4. Как происходит регистрация класса окна и создание окна.
5. Определите понятие сообщения в ОС Windows XP.
6. Назовите виды сообщений. Приведите содержание структуры MSG.
7. Как происходит создание собственного сообщения.
8. Как происходит обработка сообщений в программах.
9. Приведите типовую структуру цикла обработки сообщений.
10. Дайте сравнительную характеристику функций GetMessage() и PeekMessage()
11. Дайте сравнительную характеристику функции PostMessage() и SendMessage()
12. Предназначение и оформление диаграммы классов в нотации Буча.

13. Предназначение и оформление диаграммы объектов в нотации Буча.

14. Предназначение и оформление диаграммы состояний в нотации Буча.

15. Предназначение и оформление диаграммы модулей в нотации Буча.

16. Опишите назначение, синтаксис и использование файла ресурсов в формате .rc.

Библиографический список

1. Безруков, В.А. Программирование с использованием Win32 API: учеб. пособие / В.А.Безруков; Балт. гос. техн. ун-т. – СПб., 2008.
2. Шилд, Герберт. Программирование на C и C++ для Windows 95 : пер. с англ. / Г. Шилдт. - Киев : BHV, 1996.
3. Керниган, Брайан В. Язык программирования Си : пер. с англ. / Б. В. Керниган, Д. М. Ритчи ; ред. Вс. С. Штаркман. - Изд. 3-е, испр. - СПб. : Невский диалект, 2001.
4. Страуструп Бьерн. Язык программирования C++. Специальное издание : пер. с англ. / Б. Страуструп - С авт. изм. и доп. - М. : БИНОМ, 2004.
5. Буч, Гради. Объектно-Ориентированный Анализ и Проектирование с примерами приложений на C++ : пер. с англ. / Г. Буч. - 2-е изд. - М. : БИНОМ ; СПб. : Невский диалект, 1999.
6. Вирт, Никлаус. Алгоритмы и структуры данных : пер. с англ. / Н. Вирт. - М. : Мир, 1989
7. ГОСТ 7.32 – 2001 Отчет о научно-исследовательской работе: структура и правила оформления
8. ГОСТ 19.402-2000 ЕСПД. Описание программы. Требования к содержанию, оформлению и контролю качества
9. ГОСТ 19.401-2000 ЕСПД. Текст программы. Требования к содержанию, оформлению и контролю качества
10. ГОСТ 19.505-79 ЕСПД. Руководство оператора. Требования к содержанию и оформлению
11. ГОСТ 19.106-78 ЕСПД. Требования к программным документам, выполненным печатным способом

ОГЛАВЛЕНИЕ

Цель и содержание работы 3
Краткие сведения из теории 4
Последовательность этапов выполнения курсовой работы 29
Требования к оформлению пояснительной записки 30
Темы курсовых работ 32
Контрольные вопросы 35
Библиографический список 37