ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Б1.Б.21 Языки программирования

Направление подготовки *10.03.01 Информационная безопасность*

Профиль образовательной программы *Безопасность автоматизированных систем*

Форма обучения *очная*

СОДЕРЖАНИЕ

1. КОНСПЕКТ ЛЕКЦИЙ

1.1. Лекция № 1 (2 часа)

Тема: «Основные этапы решения задач на ЭВМ»

1.1.1. Вопросы лекции:

1) Порядок разработки программы.

2) Критерии качества программы.

1.1.2. Краткое содержание вопросов:

1) Порядок разработки программы.

При разработке программы (программного модуля) целесообразно придерживаться следующего порядка:

- изучение и проверка спецификации программы, выбор языка программирования;
- выбор алгоритма и структуры данных;
- программирование (кодирование);
- шлифовка текста;
- проверка;
- компиляция.

Первый шаг разработки программы в значительной степени представляет собой смежный контроль структуры программы снизу: изучая спецификацию модуля, разработчик должен убедиться, что она ему понятна и достаточна для разработки этого модуля. В завершении этого шага выбирается язык программирования: хотя язык программирования может быть уже предопределен для всего ПС, все же в ряде случаев (если система программирования это допускает) может быть выбран другой язык, более подходящий для реализации данного модуля (например, язык ассемблера).

На втором шаге разработки программы необходимо выяснить, не известны ли уже какие-либо алгоритмы для решения поставленной и или близкой к ней задачи. И если найдется подходящий алгоритм, то целесообразно им воспользоваться. Выбор подходящих структур данных, которые будут использоваться при выполнении программой своих функций, в значительной степени предопределяет логику и качественные показатели разрабатываемого программного модуля, поэтому его следует рассматривать как весьма ответственное решение.

На третьем шаге осуществляется построение текста модуля на выбранном языке программирования. Обилие всевозможных деталей, которые должны быть учтены при реализации функций, указанных в спецификации модуля, легко могут привести к созданию весьма запутанного текста, содержащего массу ошибок и неточностей. Искать ошибки в таком модуле и вносить в него требуемые изменения может оказаться весьма трудоемкой задачей. Поэтому весьма важно для построения текста модуля пользоваться технологически обоснованной и практически проверенной дисциплиной программирования. Впервые на это обратил внимание Дейкстра, сформулировав и обосновав основные принципы структурного программирования. На этих принципах базируются многие дисциплины программирования, широко применяемые на практике.

Следующий шаг разработки модуля связан с приведением текста модуля к завершенному виду в соответствии со спецификацией качества ПС. При программировании модуля разработчик основное внимание уделяет правильности реализации функций модуля, оставляя недоработанными комментарии и допуская некоторые нарушения требований к стилю программы. При шлифовке текста модуля он должен отредактировать имеющиеся в тексте комментарии и, возможно, включить в него дополнительные комментарии с целью обеспечить требуемые примитивы качества. С этой же целью производится редактирование текста программы для выполнения стилистических требований.

Шаг проверки модуля представляет собой ручную проверку внутренней логики модуля до начала его отладки (использующей выполнение его на компьютере).

Последний шаг разработки модуля означает завершение проверки модуля (с помощью компилятора) и переход к процессу отладки модуля.

2) Критерии качества программы.

Каждое ПС должно выполнять определенные функции, т.е. делать то, что задумано. Хорошее ПС должно обладать еще целым рядом свойств, позволяющим успешно его использовать в течении длительного периода, т.е. обладать определенным качеством. *Качество (quality*) ПС – это совокупность его черт и характеристик, которые влияют на его способность удовлетворять заданные потребности пользователей. Это не означает, что разные ПС должны обладать одной и той же совокупностью таких свойств в их наивысшей степени. Этому препятствует тот факт, что повышение качества ПС по одному из таких свойств часто может быть достигнуто лишь ценой изменения стоимости, сроков завершения разработки и снижения качества этого ПС по другим его свойствам. Качество ПС является удовлетворительным, когда оно обладает указанными свойствами в такой степени, чтобы гарантировать успешное его использование.

Совокупность свойств ПС, которая образует удовлетворительное для пользователя качество ПС, зависит от условий и характера эксплуатации этого ПС, т.е. от позиции, с которой должно рассматриваться качество этого ПС. Поэтому при описании качества ПС, прежде всего, должны быть фиксированы *критерии* отбора требуемых свойств ПС. В настоящее время *критериями качества ПС* (c*riteria of software quality*) принято считать:

* функциональность,
* надежность,
* легкость применения,
* эффективность,
* сопровождаемость,
* мобильность.

1.2. Лекция № 2, 3 (4 часа)

Тема: «Жизненный цикл программы»

1.2.1. Вопросы лекции:

1) Дружественность программы.

2) Жизненный цикл программы.

1.2.2. Краткое содержание вопросов:

1) Дружественность программы.

Легкость применения программы, в значительной степени, определяется составом и качеством пользовательской документации, а также некоторыми свойствами, реализуемыми программным путем. Следует обратить внимание на широко используемый в настоящее время подход информирования пользователя в интерактивном режиме (в процессе применения программ ПС). Такое информирование во многих случаях оказывается более удобным для пользователя, чем с помощью автономной документации, так как позволяет пользователю без какого-либо поиска вызывать необходимую информацию за счет использования контекста ее вызова. Такой подход к информированию пользователя является весьма перспективным.

Программным путем реализуются такие качества ПС как *коммуникабельность*, *устойчивость* и *защищенность*. Коммуникабельность обеспечивается соответствующей реализацией обработки исключительных ситуаций и созданием подходящего пользовательского интерфейса.

Пользовательский интерфейс представляет средство взаимодействия пользователя с ПС. При разработке пользовательского интерфейса следует учитывать потребности, опыт и способности пользователя. Поэтому потенциальные пользователи должны быть вовлечены в процесс разработки такого интерфейса. Большой эффект здесь дает его прототипирование. При этом пользователи должны получить доступ к прототипам пользовательского интерфейса, а их

оценка различных возможностей используемого прототипа должна существенно учитываться при создании окончательного варианта пользовательского интерфейса.

В силу большого разнообразия пользователей и видов ПС существует множество различных стилей пользовательских интерфейсов, при разработке которых могут использоваться разные принципы и подходы. Однако следующие важнейшие принципы следует соблюдать всегда:

- пользовательский интерфейс должен базироваться на терминах и понятиях, знакомых пользователю;
- пользовательский интерфейс должен быть единообразным;
- пользовательский интерфейс должен позволять пользователю исправлять собственные ошибки;
- пользовательский интерфейс должен позволять получение пользователем справочной информации: как по его запросу, так и генерируемой ПС.

В настоящее время широко распространены командные и графические пользовательские интерфейсы.

2) Жизненный цикл программы.

Под *жизненным циклом* ПС (*software life cycle*) понимают весь период его разработки и эксплуатации (использования), начиная от момента возникновения замысла ПС и кончая прекращением всех видов его использования. Жизненный цикл охватывает довольно сложный процесс создания и использования ПС (*software process*). Этот процесс может быть организован по-разному для разных классов ПС и в зависимости от особенностей коллектива разработчиков.

В настоящее время можно выделить 5 основных подходов к организации процесса создания и использования ПС.

- Водопадный подход. При таком подходе разработка ПС состоит из цепочки этапов. На каждом этапе создаются документы, используемые на последующем этапе. В исходном документе фиксируются требования к ПС. В конце этой цепочки создаются программы, включаемые в ПС.
- Исследовательское программирование. Этот подход предполагает быструю (насколько это возможно) реализацию рабочих версий программ ПС, выполняющих лишь в первом приближении требуемые функции. После экспериментального применения реализованных программ производится их модификация с целью сделать их более полезными для пользователей. Этот процесс повторяется до тех пор, пока ПС не будет достаточно приемлемо для пользователей. Такой подход применялся на ранних этапах развития программирования, когда технологии программирования не придавали большого значения (использовалась интуитивная технология). В настоящее время этот подход применяется для разработки таких ПС, для которых пользователи не могут точно сформулировать требования (например, для разработки систем искусственного интеллекта).
- Прототипирование. Этот подход моделирует начальную фазу исследовательского программирования вплоть до создания рабочих версий программ, предназначенных для проведения экспериментов с целью установить требования к ПС. В дальнейшем должна последовать разработка ПС по установленным требованиям в рамках какого-либо другого подхода (например, водопадного).
- Формальные преобразования. Этот подход включает разработку формальных спецификаций ПС и превращение их в программы путем корректных преобразований. На этом подходе базируется компьютерная технология (CASE-технология) разработки ПС.
- Сборочное программирование. Этот подход предполагает, что ПС конструируется, главным образом, из компонент, которые уже существуют. Должно быть некоторое хранилище (библиотека) таких компонент, каждая из которых может многократно использоваться в разных ПС. Такие компоненты называются *повторно используемыми*

(*reusable*). Процесс разработки ПС при данном подходе состоит скорее из сборки программ из компонент, чем из их программирования.

1.3. Лекция № 4, 5 (4 часа)

Тема: «Алгоритмы»

1.3.1. Вопросы лекции:

1) Способы записи алгоритма.
2) Свойства алгоритмов.
3) Блок – схемы.

1.3.2. Краткое содержание вопросов:

Информация выступает для компьютера в качестве сырья для переработки, как скажем, железная руда является сырьем для металлургического производства. Но любое сырье, прежде чем поступить на переработку, нуждается в предварительной подготовке, чтобы его переработка была наиболее эффективной. В полной мере это относится и к информации.

Предположим, к изучению какого-то явления (процесса) мы хотим привлечь вычислительную технику. Первое, что мы при этом делаем – это организуем сбор информации об интересующем нас явлении, проводим систематизацию полученной информации. После этого может быть сформулирована задача, решение которой будет способствовать обогащению нас знаниями об интересующем явлении. Для решения задачи наиболее целесообразно использовать средства вычислительной техники, при этом, как было отмечено ранее, задача должна быть подготовлена соответствующим образом.

Независимо от степени сложности решаемой задачи, процесс подготовки ее к решению на ЭВМ, как правило, состоит из следующих основных этапов:

общая постановка задачи;
математическое описание (формализация) задачи;
выбор метода решения задачи;
алгоритмизация вычислительного процесса;
составление программы на языке программирования;
отладка программы.

Общая постановка задачи является начальным этапом и предполагает формулирование задачи на содержательном (словесном) уровне. Она выполняется лицом, в интересах которого решается задача. На этом этапе осуществляется описание сущности задачи и цели ее решения (то есть, что должно быть определено в результате решения задачи). При этом на обычном русском языке записывается условие задачи, перечень постоянных и переменных исходных данных, а также порядок их поступления, перечень искомых величин и формы их представления, требуемая точность представления искомых величин и т. д. Как показывает практика, этот этап является очень ответственным и требует такой формулировки задачи, которая может быть **однозначно** истолкована.

Выбор метода решения задачи. Необходимость этого этапа обусловлена тем, что не всякое математическое описание задачи может быть пригодно для решения на ЭВМ. По этой причине **нужно выбрать и обосновать такой метод, который исходную задачу сведет к определенной последовательности элементарных действий** (арифметических и логических операций).

На практике часто одну и ту же задачу можно решить разными математическими методами. Поэтому перед лицом, подготавливающим задачу к решению на ЭВМ, стоит задача выбрать такой метод, который требует задействования наименьших ее вычислительных ресурсов. Очевидно, что успешный выбор метода решения задачи зависит от математической эрудиции лица, решающего эту задачу.

Алгоритмизация вычислительного процесса. Любая ЭВМ функционирует в соответствии с **алгоритмом**, составленным пользователем. Поэтому для решения какой-либо

прикладной задачи должен быть составлен алгоритм ее решения, то есть последовательность тех элементарных операций, которые должна выполнить ЭВМ над исходными данными и промежуточными результатами для получения конечного результата (решения задачи). Более подробно сущность алгоритма и его свойства будут изучены в третьем учебном вопросе. Пока же только заметим, что при разработке алгоритма необходимо четко представлять сущность решаемой задачи, чтобы предусмотреть все возможные варианты действий.

Создание алгоритма решения задачи представляет собой достаточно сложную творческую задачу. Конкретные навыки приходят с опытом при частом решении самых разных задач. Возможность применения накопленных знаний и навыков связана с тем, что к моменту создания алгоритма уже не имеет значения конкретное содержание задачи. В задачах из самых разных областей науки и техники, военного дела могут использоваться одни и те же (или близкие по структуре) алгоритмы. Накопленный опыт позволяет создавать алгоритмы решения задач, основываясь на принципах, применявшихся при решении совершенно иных задач.

После составления алгоритма решения задачи переходят к следующему этапу – **составлению программы** решения задачи на ЭВМ. На этом этапе алгоритм записывают на одном из алгоритмических языков программирования (BASIC, C^{++}, Pascal и т. д.). Поскольку любой из алгоритмических языков является языком высокого уровня, а ЭВМ понимает только язык низкого уровня – язык машинных кодов (двоичные коды), программа с помощью транслятора или интерпретатора переводится в машинные коды и помещается в оперативную память ЭВМ.

1.4. Лекция № 6 (2 часа)

Тема: «Программа на языке высокого уровня»

1.4.1. Вопросы лекции:

1) Стандартные типы данных.
2) Описание переменных и констант.
3) Функции и математические операторы.
4) Операторы WRITE и READ.
5) Функции для работы со строковыми переменными.

1.4.2. Краткое содержание вопросов:

Рассмотрим основные сведения об алгоритмических языках программирования, получивших наибольшее распространение среди программистов.

Fortran (**Фортран**). Это первый компилируемый язык, созданный Джимом Бэкусом в 50-е годы. Программисты, разрабатывавшие программы исключительно на ассемблере, выражали серьезное сомнение в возможности появления высокопроизводительного языка высокого уровня, поэтому основным критерием при разработке компиляторов Фортрана являлась эффективность исполняемого кода. Хотя в Фортране впервые был реализован ряд важнейших понятий программирования, удобство создания программ было принесено в жертву возможности получения эффективного машинного кода. Однако для этого языка было создано огромное количество библиотек, начиная от статистических комплексов и заканчивая пакетами управления спутниками, поэтому Фортран продолжает активно использоваться во многих организациях.

Cobol (**Кобол**). Это компилируемый язык для применения в экономической области и решения бизнес-задач, разработанный в начале 60-х годов. Он отличается большой "многословностью" – его операторы иногда выглядят как обычные английские фразы. В Коболе были реализованы очень мощные средства работы с большими объемами данных, хранящимися на различных внешних носителях. На этом языке создано очень много приложений, которые активно эксплуатируются и сегодня. Достаточно сказать, что наибольшую зарплату в США получают программисты на Коболе.

A1go1 (**Алгол**). Компилируемый язык, созданный в 1960 году. Он был призван заменить Фортран, но из-за более сложной структуры не получил широкого распространения. В 1968

году была создана версия **Алгол 68**, по своим возможностям и сегодня опережающая многие языки программирования, однако из-за отсутствия достаточно эффективных компьютеров для нее не удалось своевременно создать хорошие компиляторы. Язык Алгол сыграл большую роль в теории, но для практического программирования сейчас почти не используется.

Pascal (**Паскаль**). Язык Паскаль, созданный в конце 70-х годов основоположником множества идей современного программирования Никлаусом Виртом и названный в честь ученого Блеза Паскаля, во многом напоминает Алгол, но в нем ужесточен ряд требований к структуре программы и имеются возможности, позволяющие успешно применять его при создании крупных программных проектов. Изначально Паскаль был задуман как язык программирования для студентов. Он и сейчас является основным языком программирования, используемым в качестве учебного, во многих высших учебных заведениях. На базе языка Паскаль созданы несколько более мощных языков (Модула, Ада, Дельфи).

BASIC (**Бейсик**). Для этого языка имеются и компиляторы, и интерпретаторы, а по популярности он занимает первое место в мире. Бейсик был создан в 1964 г. Томасом Куртом и Джоном Кемени как язык для начинающих программистов, облегчающий написание простых программ. В настоящее время разработано несколько различных версий языка Бейсик, таких как GWBASIC, QBASIC, Pro-BASIC, Turbo-BASIC, Visual-BASIC и др.

C (**Си**). Данный язык был создан в лаборатории Be11 и первоначально не рассматривался как массовый. Он планировался для замены ассемблера, чтобы иметь возможность создавать столь же эффективные и компактные программы, и в то же время не зависеть от конкретного типа процессора.

Си во многом похож на Паскаль и имеет дополнительные средства для прямой работы с памятью (**указатели**). На этом языке в 70-е годы написано множество прикладных и системных программ и ряд известных операционных систем (например, Unix). В настоящее время Си широко используется при создании системного программного обеспечения.

PL/1 (**ПЛ/1**). В середине 60-х годов компания **1BM** решила взять все лучшее из языков Фортран, Кобол и Алгол и воплотить в одном языке программирования. В результате в 1964 году на свет появился новый компилируемый язык программирования, который получил название **Programming Language One**. В этом языке было реализовано множество уникальных решений, полезность которых удается оценить только спустя 33 года, в эпоху крупных программных систем. По своим возможностям **ПЛ/1** значительно мощнее многих других языков (Си, Паскаля). Например, в **ПЛ/1** присутствует уникальная возможность указания точности вычислений – ее нет даже у **Си++ и Явы**. В настоящее время язык **ПЛ/1** почти не используется.

Ada (**Ада**). Назван по имени леди Огасты Ады Лавлейс, дочери английского поэта Байрона и его отдаленной родственницы Анабеллы Милбэнк. В 1979 году сотни экспертов Министерства обороны США отобрали из 17 вариантов именно этот язык, разработанный небольшой группой под руководством Жана Ихбиа. Он удовлетворил на то время все требования Пентагона, а к сегодняшнему дню в его развитие вложены десятки миллиардов долларов. Структура самого языка похожа на Паскаль. В нем имеются средства строгого разграничения доступа к различным уровням спецификаций, доведена до предела мощность управляющих конструкций.

1.5. Лекция № 7, 8 (4 часа)

Тема: «Представления основных структур программирования»

1.5.1. Вопросы лекции:

1) Оператор безусловного перехода.
2) Оператор условного перехода.
3) Логические операторы. Таблицы истинности.
4) Оператор CASE.
5) Цикл FOR.
6) Цикл WHILE. Цикл REPEAT…UNTIL.
7) Массивы.

1.5.2. Краткое содержание вопросов:

Оператор перехода указывает, что дальше программа должна выполняться, начиная с оператора, помеченного меткой, которая записана в этом операторе перехода.

Оператор перехода имеет вид: goto <метка>. Здесь goto - зарезервированное слово (перейти на ... [метку]).

Метка в Турбо Паскале - это произвольный идентификатор, позволяющий именовать (помечать) некоторый оператор программы и таким образом ссылаться на него. Допускается в качестве меток (в том числе) использовать целые числа без знака.

Например:

goto 2341, goto 23, goto 1, goto bl1, goto mnk3, goto n56.

Любая метка должна быть описана в разделе описания меток, который обычно располагается в программе до начала раздела операторов:

label <список меток>.

Например: label 2341; label 1, 2; label bl1, 18, mnk.

В программе метка записывается перед оператором, на который осуществляется переход. Метка отделяется от оператора двоеточием. Между меткой и оператором можно включать один или несколько пробелов, например:

```
1: writeln('Число не содержит цифру 2');
```

Оператор можно помечать несколькими метками, которые в этом случае отделяются друг от друга двоеточием.

```
1: 25:  a := b div 10;
```

Итак, оператор перехода изменяет последовательность выполнения операторов - они выполняются не в том порядке, как написаны в тексте программы.

Оператор if … then … else

Для реализации принимаемых решений на языке Паскаль могут быть использованы операторы if (если), then (тогда) и else (иначе).

После оператора if записывается условие, а после операторов then и else - команды, которые необходимо выполнить.

Перед оператором else точка с запятой не ставится.

Если после оператора then записано несколько команд, в нём устанавливаются операторные скобки:

```
begin
  .....
end;
```

Такие же операторные скобки в необходимых случаях могут быть использованы и в операторе else.

1.6. Лекция № 9, 10 (4 часа)

Тема: «Структурированный тип данных»

1.6.1. Вопросы лекции:

1) Ошибки программирования.

2) Помощь в системе программирования.

3) Компиляторы и интерпретаторы.

1.6.2. Краткое содержание вопросов:

Ошибки в программировании бывают двух типов: синтаксические и логические. Синтаксические ошибки это неправильное написание ключевых слов и символов языка программирования. Все виды синтаксических ошибок могут быть найдены компьютером в 99 % случаев.

Логические ошибки связаны с неправильной программной реализацией логики алгоритма. В 99 % случаев компьютер будет выполнять программу, но результат работы будет неправильный.

Примеры синтаксических ошибок в языке Паскаль и их исправление:

1) Не поставлена точка с запятой. После пуска программы, нажатием клавиш <Ctrl>+<F9>, в верхней строке экрана появится написанное красным цветом сообщение:

Error 85: ";" expected.

(Ошибка 85: ";" отсутствует.)

Редактор установит курсор на следующий символ после пропущенного знака. После нажатия любой клавиши, сообщение об ошибке исчезает, и редактор переходит в режим вставки. Надо подвести курсор к нужному месту, поставить точку с запятой - “;” и продолжить работу.

2) В описании переменных не записана переменная, а в программе она присутствует. После пуска программы, будет выдано сообщение:

Error 3: Unknown identifier.

(Ошибка 3: Неизвестный идентификатор.)

Курсор будет установлен на эту переменную. Надо исправить ошибку, т.е. записать переменную в раздел описаний переменных и продолжить работу.

3) Не поставлена точка после оператора end в конце программы. Сообщение компилятора будет таким:

Error 10: Unexpected end of file.

(Ошибка 10: Неправильный конец файла.),

Курсор установится на букву "e" в слове "end". Надо поставить точку и снова выполнить программу.

1.7. Лекция № 11 (2 часа)

Тема: «Понятие объекта и класса»

1.7.1. Вопросы лекции:

1) Подпрограммы.

2) Программирование рекурсивных алгоритмов.

1.7.2. Краткое содержание вопросов:

Программист должен видеть в целом программу, которая решает какую-то задачу, а потом разбивает ее на отдельные части, составляет на выбранном языке программирования эти части программы, объединяет их в единое целое и получает программу.

Итак, весь творческий процесс можно разбить (разумеется, чисто условно) на следующие этапы:

1) основная идея решения задачи;

2) общая конструкция программы;

3) выделение отдельных, элементарных частей программы;

4) практическая реализация на языке программирования этих частей программы;

5) объединение их в единую программу.

Такой процесс программирования называют структурным или нисходящим. Более подробно с этим процессом мы познакомимся позже, когда изучим хотя бы основы языка программирования, но об отдельных частях, "кирпичиках", составляющих программу узнаем на этом занятии.

Подпрограммой называется группа операторов, к которой обращаются из основной программы несколько раз. Иногда это может быть 2, 3 раза, а очень часто, каждый раз из выполняемого цикла основной программы.

Вполне понятно, что писать несколько раз одинаковые группы операторов трудно, проделывается много "*технической*" работы, а в некоторых случаях просто невозможно (если обращаться приходиться каждый раз при выполнении цикла).

Для облегчения такой работы и созданы подпрограммы.

Использование подпрограмм позволяет:

1) сделать основную программу более наглядной и компактной;

2) уменьшить объем используемой памяти ЭВМ;

3) сократить время отладки программы.

На языке Паскаль *подпрограммы* бывают двух видов, - это процедуры и функции.

Процедуры

Рассмотрим следующий простой пример, с помощью которого попробуем разобраться в конструкции процедур на Паскале.

Пример. Составить программу, которая бы проверяла, являются ли три числа взаимно простыми.

Мы знаем, что числа называются взаимно простыми, если их наибольший общий делитель (НОД) равен 1. Значит, для решения этой задачи нам придется дважды находить НОД чисел. Если заданы три числа: a, b, c, то найти НОД(a, b), а затем найти НОД(НОД(a, b), c).

Дважды писать операторы для нахождения НОД нам не хочется, поэтому оформим операторы для НОД в виде процедуры.

Посмотрите, как это будет выглядеть в *программе*:

```
Program Problem1;

  var
    a, b, c, k : integer;
{-------------------------------------------------------------------------------}
  Procedure nod(a, b : integer; var n : integer);
    var
      r : integer;
   begin
     repeat
       r := a mod b;
       a := b; b := r
     until b = 0;
     n := a
   end;
{-------------------------------------------------------------------------------}
 begin
```

```
  write('Введите три натуральных числа '); readln(a, b, c);
  nod(a, b, k);
  a := k; b := c;
  nod(a, b, k);
    if k = 1 then writeln('Числа взаимно простые')
             else writeln('Числа не взаимно простые')
end.
```

1.8. Лекция № 12 (2 часа)

Тема: «Свойства объектов, методы, события»

1.8.1. Вопросы лекции:

1) Типы данных, определяемые пользователем.
2) Структуры.
3) Файловый тип.

1.8.2. Краткое содержание вопросов:

Запись — это структурированный тип данных, состоящий из фиксированного числа компонентов одного или нескольких типов. Определение типа записи начинается идентификатором record и заканчивается зарезервированным словом end. Между ними заключен список компонентов, называемых **полями,** с указанием идентификаторов полей и типа каждого поля.

Формат:

```
type
<имя типа> = record
        <Идентификатор поля>:<тип компонента>;
        . . .
                <Идентификатор поля>:<тип компонента>
        end;
                                var
<идентификатор,...> : <имя типа>;
Пример:
type
                                Car = record
        Number:integer;      {Номер}
Marka:string[20];    {Марка автомобиля}
FIO:string[40];              {Фамилия, инициалы владельца}
Address:string[60]   {Адрес владельца}
end;
                                var
                                M, V : Car;
```

В данном примере запись Car содержит четыре компонента: номер, название марки машины, фамилию владельца и его адрес. Доступ к полям записи осуществляется через переменную типа "запись". В нашем случае это переменные M и V типа Car.

Идентификатор поля должен быть уникален только в пределах записи, однако во избежание ошибок лучше делать его уникальным в пределах всей программы. Объем памяти, необходимый для записи, складывается из длин полей.

Значения полей записи могут быть использованы в выражениях. Имена отдельных полей не применяются по аналогии с идентификаторами переменных, поскольку может быть несколько записей одинакового типа. Обращение к значению поля осуществляется с помощью идентификатора переменной и идентификатора поля, разделенных точкой. Такая комбинация

называется составным именем. Например - чтобы получить доступ к полям записи Саг, надо записать:

1.9. Лекция № 13, 14 (4 часа)

Тема: «Иерархия, наследование, полиморфизм»

1.9.1. Вопросы лекции:

1) Статические и динамические переменные.
2) Типизированные и нетипизированные указатели.
3) Управление динамической памятью.
4) Списки: основные виды и способы реализации.

1.9.2. Краткое содержание вопросов:

Все переменные, объявленные в программе, размещаются в одной непрерывной области оперативной памяти, которая называется сегментом данных. Длина сегмента данных определяется архитектурой микропроцессоров 80х86 и составляет 65536 байт, что может вызвать известные затруднения при обработке больших массивов данных. С другой стороны, объем памяти ПК (обычно не менее 640 Кбайт) достаточен для успешного решения задач с большой размерностью данных. Выходом из положения может служить использование так называемой динамической памяти.

Динамическая память - это оперативная память ПК, предоставляемая программе при ее работе, за вычетом сегмента данных (64 Кбайт), стека (обычно 16 Кбайт) и собственно тела программы. Размер динамической памяти можно варьировать в широких пределах (см. прил.1). По умолчанию этот размер определяется всей доступней памятью ПК и, как правило, составляет не менее 200...300 Кбайт.

Динамическая память - это фактически единственная возможность обработки массивов данных большой размерности. Многие практические задачи трудно или невозможно решить без использования динамической памяти. Такая необходимость возникает, например, при разработке систем автоматизированного проектирования (САПР): размерность математических моделей, используемых в САПР, может значительно отличаться в разных проектах; статическое (т.е. на этапе разработки САПР) распределение памяти в этом случае, как правило, невозможно. Наконец, динамическая память широко используется для временного запоминания данных при работе с графическими и звуковыми средствами ПК.

Динамическое размещение данных означает использование динамической памяти непосредственно при работе программы. В отличие от этого статическое размещение осуществляется компилятором Турбо Паскаля в процессе компиляции программы. При динамическом размещении заранее не известны ни тип, ни количество размещаемых данных, к ним нельзя обращаться по именам, как к статическим переменным.

Адреса и указатели

Оперативная память ПК представляет собой совокупность элементарных ячеек для хранения информации - байтов, каждый из которых имеет собственный номер. Эти номера называются адресами, они позволяют обращаться к любому байту памяти.

Турбо Паскаль предоставляет в распоряжение программиста гибкое средство управления динамической памятью - так называемые указатели. Указатель - это переменная, которая в качестве своего значения содержит адрес байта памяти.

В ПК адреса задаются совокупностью двух шестнадцатиразрядных слов, которые называются сегментом и смещением. Сегмент - это участок памяти, имеющий длину 65536 байт (64 Кбайт) и начинающийся с физического адреса, кратного 16 (т.е. О, 16, 32, 48 и т.д.). Смещение указывает, сколько байт от начала сегмента необходимо пропустить, чтобы обратиться к нужному адресу.

Адресное пространство ПК составляет 1 Мбайт (речь идет о так называемой стандартной памяти ПК; на современных компьютерах с процессорами 80386 и выше адресное пространство

составляет 4 Гбайт, однако в Турбо Паскале нет средств, поддерживающих работу с дополнительной памятью; при использовании среды Borland Pascal with Objects 7.0 такая возможность имеется). Для адресации в пределах 1 Мбайта нужно 20 двоичных разрядов, которые получаются из двух шестнадцатиразрядных слов (сегмента и смещения) следующим образом: содержимое сегмента смещается влево на 4 разряда, освободившиеся правые разряды заполняются нулями, результат складывается с содержимым смещения.

1.10. Лекция № 15 (2 часа)

Тема: «Графическая среда разработки»

1.10.1. Вопросы лекции:

1) Способы конструирования программ.

2) Модульные программы.

1.10.2. Краткое содержание вопросов:

Каскадная модель.

Основной характеристикой каскадной модели является разбиение всей разработки на этапы, причем переход с одного этапа на следующий происходит только после того, как будет полностью завершена работа на текущем (рис. 1). Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

Положительные стороны применения каскадного подхода заключаются в следующем:

на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;

выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

Каскадный подход хорошо зарекомендовал себя при построении ПС, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, с тем чтобы предоставить разработчикам свободу реализовать их как можно лучше с технической точки зрения. В эту категорию попадают сложные расчетные системы, системы реального времени и другие подобные задачи. Однако, в процессе использования этого подхода обнаружился ряд его недостатков, вызванных прежде всего тем, что реальный процесс создания ПО никогда полностью не укладывался в такую жесткую схему. В процессе создания ПО постоянно возникала потребность в возврате к предыдущим этапам и уточнении или пересмотре ранее принятых решений. В результате реальный процесс создания ПО принимал следующий вид (рис. 2.):

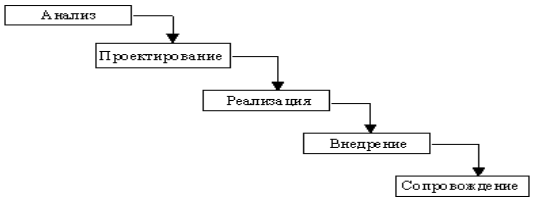

Рис. 1. - Каскадная схема

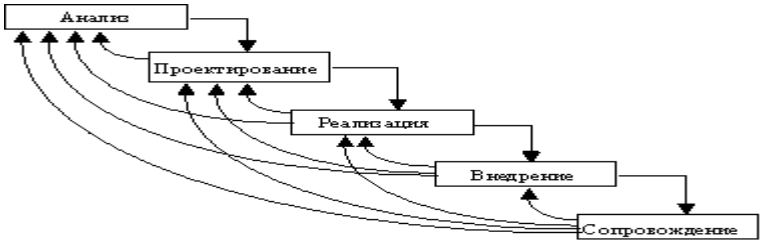

Рисунок 2. - Реальный процесс разработки

Основным **недостатком** каскадного подхода является существенное запаздывание с получением результатов. Согласование результатов с пользователями производится только в

точках, планируемых после завершения каждого этапа работ, требования к ПС «заморожены» в виде технического задания на все время ее создания. Таким образом, пользователи могут внести свои замечания только после того, как работа над системой будет полностью завершена. В случае неточного изложения требований или их изменения в течение длительного периода создания ПО, пользователи получают систему, не удовлетворяющую их потребностям. Модели (как функциональные, так и информационные) автоматизируемого объекта могут устареть одновременно с их утверждением.

1.11. Лекция № 16 (2 часа)

Тема: «Визуальное проектирование графического интерфейса»

1.11.1. Вопросы лекции:

1) Понятие правильной программы.

2) Надежность программного средства.

1.11.2. Краткое содержание вопросов:

Понятие правильной программы.

Под «программой» часто понимают правильную программу, т.е. программу, не содержащую ошибок. Однако, понятие ошибки в программе трактуется в среде программистов неоднозначно. Согласно Майерсу будем считать, что в программе имеется *ошибка*, если она не выполняет того, что разумно ожидать от нее пользователю. «Разумное ожидание» пользователя формируется на основании документации по применению этой программы. Следовательно, понятие ошибки в программе является существенно не формальным. В ПС программы и документация взаимно увязаны, образуют некоторую целостность. Поэтому правильнее говорить об ошибке не в программе, а в ПС в целом: будем считать, что в ПС имеется *ошибка* (*software error*), если оно не выполняет того, что разумно ожидать от него пользователю. В частности, разновидностью ошибки в ПС является несогласованность между программами ПС и документацией по их применению. Выделяется в отдельное понятие частный случай ошибки в ПС, когда программа не соответствует своей функциональной спецификации (описанию, разрабатываемому на этапе, предшествующему непосредственному программированию). Такая ошибка в указанной работе называется *дефектом программы*. Однако выделение такой разновидности ошибки в отдельное понятие вряд ли оправданно, так как причиной ошибки может оказаться сама функциональная спецификация, а не программа.

Так как задание на ПС обычно формулируется не формально, а также из-за того, что понятия ошибки в ПС не формализовано, то нельзя доказать формальными методами (математически) правильность ПС. Нельзя показать правильность ПС и тестированием, тестирование может лишь продемонстрировать наличие в ПС ошибки. Поэтому понятие правильной ПС неконструктивно в том смысле, что после окончания работы над созданием ПС мы не сможем убедиться, что достигли цели.

Надежность программного средства.

Альтернативой правильного ПС является *надежное* ПС. *Надежность* (*reliability*) ПС – это его способность безотказно выполнять определенные функции при заданных условиях в течение заданного периода времени с достаточно большой вероятностью. При этом под *отказом* в ПС понимают проявление в нем ошибки. Таким образом, надежное ПС не исключает наличия в нем ошибок – важно лишь, чтобы эти ошибки при практическом применении этого ПС в заданных условиях проявлялись достаточно редко. Убедиться, что ПС обладает таким свойством можно при его испытании путем тестирования, а также при практическом применении. Таким образом, фактически мы можем разрабатывать лишь надежные, а не правильные ПС.

ПС может обладать различной степенью надежности. Как измерять эту степень? Так же как в технике, степень надежности можно характеризовать вероятностью работы ПС без отказа

в течение определенного периода времени. Однако в силу специфических особенностей ПС определение этой вероятности наталкивается на ряд трудностей по сравнению с решением этой задачи в технике.

При оценке степени надежности ПС следует также учитывать последствия каждого отказа. Некоторые ошибки в ПС могут вызывать лишь некоторые неудобства при его применении, тогда как другие ошибки могут иметь катастрофические последствия, например, угрожать человеческой жизни. Поэтому для оценки надежности ПС иногда используют дополнительные показатели, учитывающие стоимость (вред) для пользователя каждого отказа.

1.12. Лекция № 17, 18 (4 часа)

Тема: «Библиотеки визуальных компонентов»

1.12.1. Вопросы лекции:

1) Графическая среда разработки

2) Библиотека визуальных компонентов Lazarus

1.12.2. Краткое содержание вопросов:

Графическая среда разработки

Lazarus — бесплатная и свободная графическая среда разработки программного обеспечения на языке Object Pascal для компилятора **Free Pascal Compiler** (FPC), также свободного программного обеспечения. Оба продукта распространяются под лицензией GPL (*General Public License – универсальная общественная лицензия GNU, Универсальная общедоступная лицензия GNU* или *Открытое лицензионное соглашение GNU*).

Object Pascal (Объектный Паскаль) – язык программирования, разработанный в фирме Apple Computer в 1986 году группой Ларри Теслера, который консультировался с Никлаусом Виртом. Произошел от более ранней объектно–ориентированной версии языка Паскаль.

Lazarus является кросплатформенной средой разработки, т.е. позволяет создавать программы в графическом окружении, практически неизменном, для широкого класса операционных систем: Linux, FreeBSD, Mac OS X, Microsoft Windows, Android. Для создания исполняемой программы достаточно снова собрать ее в соответствующей операционной системе. Является практически единственной простой средой разработки, в которой можно создавать приложения программистам знакомым с Delphi.

Среда разработки базируется на библиотеке визуальных компонентов *Lazarus Component Library* (LCL), которая содержит достаточное число компонент, позволяющих создавать формы при помощи визуального проектирования графического интерфейса пользователя (GUI, англ. *graphical user interface).*В Lazarus также возможно разрабатывать консольные приложения и динамически подгружаемые библиотеки: в Windows dynamic-link library, или DLL; в OS X – библиотеки dylib (dynamic shared library); в Linux –библиотеки .so (shared object library). Динамические библиотеки используются для разработки других программ.

Библиотека визуальных компонентов

Библиотека визуальных компонентов(Visual Component Library, VCL) содержит большое количество классов, предназначенных для быстрой разработки приложений. Библиотека написана на Object Pascal. В VCL содержатся невизуальные и визуальные 21 компоненты, а также другие классы, начиная с абстрактного класса *TObject*. При этом все компоненты являются классами, но не все классы являются компонентами.

Все классы VCL расположены на определенном уровне иерархии и образуют дерево (иерархию) классов. Знание происхождения объекта оказывает значительную помощь при его изучении, так как потомок наследует все элементы объекта-родителя. Так, если свойство *caption* принадлежит классу *TControi*, то это свойство будет и у его потомков, например у классов TButton и TCheckBox и у компонентов — кнопки Button и независимого переключателя CneckBox соответственно.

2. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ПРОВЕДЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

2.1. Практическое занятие №1, 2 (4 часа).
Тема: «Основные этапы решения задач на ЭВМ».
2.1.1. Вопросы к занятию:
1) Основные этапы решения задач на ЭВМ.
2.1.2. Краткое описание проводимого занятия:
2.1.2.1. Ответы на вопросы семинарского (практического) занятия.
2.1.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Выберите правильную последовательность этапов решения задачи на ЭВМ
+ a) Постановка задачи, Алгоритм задачи, Блок-схема, Программа
  b) Постановка задачи, Блок-схема, Алгоритм задачи, Программа
  c) Постановка задачи, Алгоритм задачи, Программа, Блок-схема
  d) Алгоритм задачи, Постановка задачи, Блок-схема, Программа
2. Как обозначается блок условие в блок-схеме
+ a) ромб
  b) параллелограмм
  c) прямоугольник
  d) эллипс
3. В чем выражается массовость алгоритма
+ a) в его пригодности для решения задачи с любыми исходными данными
  b) в его пригодности для решения нескольких задач определенного класса
  c) в его записи на нескольких языках программирования
  d) в решении одной и той же задачи несколькими способами

2.2. Практическое занятие № 3, 4, 5, 6 (8 часов).
Тема: «Жизненный цикл программы».
2.2.1. Вопросы к занятию:
1) Дружественность программы.
2) Жизненный цикл программы.
2.2.2. Краткое описание проводимого занятия:
2.2.2.1. Ответы на вопросы семинарского (практического) занятия.
2.2.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Какой подход к разработке программных средств называют водопадным
+ a) на каждом этапе создаются документы, используемые на последующем этапе
  b) быстрая реализация рабочих версий программы
  c) быстрая реализация рабочих версий программы, в дальнейшем разработка в рамках другого подхода
  d) разработка формальных спецификаций и превращение в программы путем корректных преобразований
2. Какой подход к разработке программных средств предполагает конструирование программ из уже существующих компонент
+ a) сборочное программирование
  b) водопадный подход
  c) исследовательское программирование
  d) прототипирование
3. Какие стадии жизненного цикла программных средств различают в водопадном подходе

+ a) разработка, производство программных изделий, эксплуатация
  b) внешнее описание, производство программных изделий, применение
  c) разработка, конструирование, эксплуатация
  d) конструирование, производство программных изделий, сопровождение

2.3. Практическое занятие № 7, 8, 9, 10 (8 часов).
Тема: «Алгоритмы».
2.3.1. Вопросы к занятию:
1) Постановка задачи и спецификация программы.
2.3.2. Краткое описание проводимого занятия:
2.3.2.1. Ответы на вопросы семинарского (практического) занятия.
2.3.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Что представляет собой блок-схема алгоритма
+ a) совокупность геометрических фигур, соединенных между собой
  b) совокупность алгоритмов, соединенных между собой
  c) совокупность геометрических фигур, соединенных в одну
  d) совокупность прямоугольников, соединенных между собой
2. Как обозначается блок условие в блок-схеме
+ a) ромб
  b) параллелограмм
  c) прямоугольник
  d) эллипс
3. В чем выражается массовость алгоритма
+ a) в его пригодности для решения задачи с любыми исходными данными
  b) в его пригодности для решения нескольких задач определенного класса
  c) в его записи на нескольких языках программирования
  d) в решении одной и той же задачи несколькими способами

2.4. Практическое занятие № 11, 12 (4 часа).
Тема: «Программа на языке высокого уровня».
2.4.1. Вопросы к занятию:
1) Постановка задачи и спецификация программы.
2.4.2. Краткое описание проводимого занятия:
2.4.2.1. Ответы на вопросы семинарского (практического) занятия.
2.4.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. На первом этапе проектирования программ
+1) разрабатываются алгоритмы, задаваемые спецификациями;
  2) создается система ввода данных, вывода результатов;
  3) выбирается математический аппарат, кодируется программа;
  4) формируется общая модульная структура программы.
2. Основное преимущество графического режима для интерфейса пользователя
+1) хорошая наглядность;
  2) большой выбор шрифтов;
  3) различная цветовая гамма;
  4) высокая разрешающая способность.
3. Наибольший эффект при использования параметра-константы достигается тогда, когда в процедуру передаются
+1) массивы и записи, т.к. они занимают значительную часть стека;
  2) длинные переменные целого типа, которые занимают весь стек;

3) переменные вещественного типа, поскольку этот тип занимает значительное место в памяти;
4) переменные целого типа, т.к. упрощается передача значений.

2.5. Практическое занятие № 13, 14, 15, 16 (8 часов).
Тема: «Представления основных структур программирования».
2.5.1. Вопросы к занятию:
1) Программа на языке высокого уровня.
2) Типы данных в языке высокого уровня.
2.5.2. Краткое описание проводимого занятия:
2.5.2.1. Ответы на вопросы семинарского (практического) занятия.
2.5.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Какие языки называют языками программирования высокого уровня
+ a) имитирующие естественные языки
b) имитирующие машинный код
c) имитирующие английский язык
d) имитирующие высокоуровневый язык
2. Что такое оператор
+ a) фраза языка, однозначно определяющая трактуемый этап обработки данных
b) ключевое слово языка, однозначно определяющее трактуемый этап обработки данных
c) подпрограмма, однозначно определяющая трактуемый этап обработки данных
d) имя переменной или идентификатор переменной
3. Что означает идентификатор языка программирования
+ a) имя программного объекта
b) порядковый номер программного объекта
c) принцип работы программного объекта
d) принцип работы функции

2.6. Практическое занятие № 17, 18, 19, 20 (8 часов).
Тема: «Структурированный тип данных».
2.6.1. Вопросы к занятию:
1) Функции и математические операторы.
2) Операторы WRITE и READ.
3) Функции для работы со строковыми переменными.
2.6.2. Краткое описание проводимого занятия:
2.6.2.1. Ответы на вопросы семинарского (практического) занятия.
2.6.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Какая последовательность операций представлена в нарушении приоритета?
1) вызов функции; деление нацело; сложение;
2) выражение, заключенное в скобки; возведение в степень;
3) остаток от деления нацело; вычитание;
+4) деление нацело; умножение.
2. Какие типы данных занимают память менее 4 байт?
+1) Byte
+2) Boolean
+3) Integer
4) Single
5) Double
6) Long Integer

3. Какая функция возвращает число, содержащееся в строке string, как числовое значение?
+1) Val (String)
2) Str (number)
3) CStr (Выражение)
4) CInt (Выражение)

2.7. Практическое занятие № 21, 22 (4 часа).
Тема: «Понятие объекта и класса».
2.7.1. Вопросы к занятию:
1) Оператор безусловного перехода.
2) Оператор условного перехода.
3) Логические операторы.
4) Таблицы истинности.
5) Оператор CASE.
2.7.2. Краткое описание проводимого занятия:
2.7.2.1. Ответы на вопросы семинарского (практического) занятия.
2.7.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Какой из знаков обозначает "не равно"
+ a) <>
b) ><
c) =>
d) <=
2. Какое из слов обозначает логическое "ИЛИ"
+ a) OR
b) XOR
c) AND
d) NOT
3. Какое из слов обозначает логическое исключающее "ИЛИ"
+ a) XOR
b) OR
c) AND
d) NOT

2.8. Практическое занятие № 23, 24 (4 часа).
Тема: «Свойства объектов, методы, события».
2.8.1. Вопросы к занятию:
1) Цикл FOR.
2) Цикл WHILE.
3) Цикл REPEAT…UNTIL.
4) Одномерные массивы.
5) Двумерные массивы.
2.8.2. Краткое описание проводимого занятия:
2.8.2.1. Ответы на вопросы семинарского (практического) занятия.
2.8.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Как правильно указывается имя элемента двумерного массива
+ a) A[3,5]
b) A(3,5)
c) A[3.5]
d) A(3.5)

2. Как правильно оформляется оператор цикла "FOR"
+ a) FOR переменная:= нач.знач. TO кон.знач. DO программный блок;
  b) FOR переменная:= нач.знач. TO кон.знач. программный блок;
  c) FOR переменная:= нач.знач. DO кон.знач. TO программный блок;
  d) FOR переменная:= нач.знач. TO DO кон.знач. программный блок;
3. Как правильно описывается массив переменных
+ a) имя массива:array[мин. номер . . макс. номер] of тип;
  b) имя массива:array[мин. номер . . макс. номер]:тип;
  c) имя массива:тип of array[мин. номер . . макс. номер];
  d) имя массива:array[мин. номер , макс. номер] of тип;

2.9. Практическое занятие № 25, 26, 27, 28 (8 часов).
Тема: «Иерархия, наследование, полиморфизм».
2.9.1. Вопросы к занятию:
1) Ошибки программирования.
2) Помощь в системе программирования.
3) Компиляторы и интерпретаторы.
2.9.2. Краткое описание проводимого занятия:
2.9.2.1. Ответы на вопросы семинарского (практического) занятия.
2.9.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Переменные, объявленные внутри процедуры или функции, называются
+1) локальными;
  2) процедурными;
  3) частными;
  4) внутренними.
2. Переменная в программировании полностью характеризуется:
  1) именем;
+2) именем, значением и типом;
  3) именем и типом;
  4) именем и значением;
  5) значением.
3. Что такое синтаксис языка программирования
+ a) система правил, определяющих допустимые конструкции языка программирования из букв алфавита
  b) система правил, определяющих допустимые конструкции языка программирования из английских букв
  c) система правил, определяющих допустимые ключевые слова
  d) система правил, определяющих допустимые имена переменных

2.10. Практическое занятие № 29, 30 (4 часа).
Тема: «Графическая среда разработки».
2.10.1. Вопросы к занятию:
1) Подпрограммы.
2) Рекурсивные алгоритмы.
2.10.2. Краткое описание проводимого занятия:
2.10.2.1. Ответы на вопросы семинарского (практического) занятия.
2.10.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Специальным образом оформленный блок программного кода, который выполняет действия в пределах своего блока и возвращает значение - это ____.
ОТВЕТ: Функция

ОТВЕТ: функция
2. Именованная часть кода, выполняющая определенные действия: вывод текста, выполнение арифметических действий, проигрывание видеофрагмента – это _____.
ОТВЕТ: Процедура
ОТВЕТ: процедура
3. Оператор, который объявляет глобальную переменную на уровне проекта, которая доступна для всех его модулей.
ОТВЕТ: Public

2.11. Практическое занятие № 31, 32, 33 (6 часов).
Тема: «Визуальное проектирование графического интерфейса».
2.11.1. Вопросы к занятию:
1) Нахождение максимального и минимального элемента массива.
2) Ранжирование массивов.
2.11.2. Краткое описание проводимого занятия:
2.11.2.1. Ответы на вопросы семинарского (практического) занятия.
2.11.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. В каком виде сортировки массивов элементы попарно сравниваются с соседними и при необходимости меняются местами
+ a) Пузырьковая сортировка
b) Классическая сортировка
c) Сортировка включениями
d) Быстрая сортировка
2. В каком виде сортировки массивов находится минимальный или максимальный элемент и переставляется на первое место
+ a) Классическая сортировка
b) Пузырьковая сортировка
c) Сортировка включениями
d) Быстрая сортировка
3. В каком виде сортировки массивов происходит разделение массива на отсортированную и неотсортированную части
+ a) Сортировка включениями
b) Пузырьковая сортировка
c) Классическая сортировка
d) Быстрая сортировка

2.12. Практическое занятие № 34, 35, 36, 37 (8 часов).
Тема: «Библиотеки визуальных компонентов».
2.12.1. Вопросы к занятию:
1) Способы конструирования программ.
2) Модульные программы.
2.12.2. Краткое описание проводимого занятия:
2.12.2.1. Ответы на вопросы семинарского (практического) занятия.
2.12.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Подход к разработке программного комплекса, при котором он разбивается на программные модули (программы), образующие многоуровневую структуру, - это
+1) нисходящая разработка;
2) структурное программирование;
3) сквозной контроль;
4) макетирование.

2. Программный комплекс - это сложная программа, состоящая из
+1) многих модулей и подпрограмм;
2) различных текстовых процессоров;
3) оверлейных процедур;
4) операторов, реализующих вычисления над комплексными числами.
3. Стиль программирования - это
+1) набор приемов, позволяющих получить легко читаемую, эффективную программу;
2) синтаксическая проверка программы на наличие ошибок;
3) использование объектно-ориентированного программирования;
4) массовое использование комментариев к каждому оператору программы.