Министерство образования и науки РФ
Федеральное государственное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

ПРОГРАММИРОВАНИЕ И АЛГОРИТМИЗАЦИЯ

сборник учебно-методических материалов
для направления подготовки 15.03.04 Автоматизация технологических процессов и производств

Благовещенск, 2017

Печатается по решению
Редакционно-издательского совета
Энергетического факультета
Амурского государственного университета

Составитель: Рыбалев А.Н.

Программирование и алгоритмизация: сборник учебно-методических материалов для направления подготовки 15.03.04 Автоматизация технологических процессов и производств. – Благовещенск: Амурский гос. ун-т, 2017. – 26 с.

Содержание

ВВЕДЕНИЕ

Целью дисциплины является обучение студентов основам прикладного программирования и алгоритмизации, а также их подготовка к изучению будущих курсов, связанных с электронно-вычислительной техникой, программированием, моделированием и т.д.

Задачи дисциплины:

изучение структуры и состава микропроцессорной системы, ее системного и прикладного программного обеспечения;

изучение основ программирования на языке ассемблера;

освоение языка программирования высокого уровня;

изучение процедурного и объектно-ориентированного подходов в программировании;

изучение структур данных и алгоритмов обработки данных;

изучение основ проектирования программных систем;

изучение систем визуального программирования графических интерфейсов;

изучение основ офисного программирования.

В результате освоения дисциплины обучающийся должен демонстрировать следующие результаты образования:

1) Знать:

основные элементы микропроцессорной системы, назначение и состав системного и прикладного программного обеспечения;

синтаксис и семантику алгоритмического языка программирования, принципы и методологию построения алгоритмов программных систем;

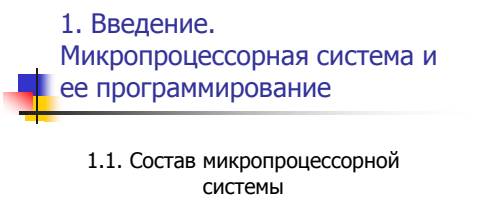
принципы структурного и модульного программирования с поддержкой жизненного цикла программ, а также объектно-ориентированного программирования.

2) Уметь проектировать простые программные алгоритмы и реализовывать их с помощью современных средств программирования.

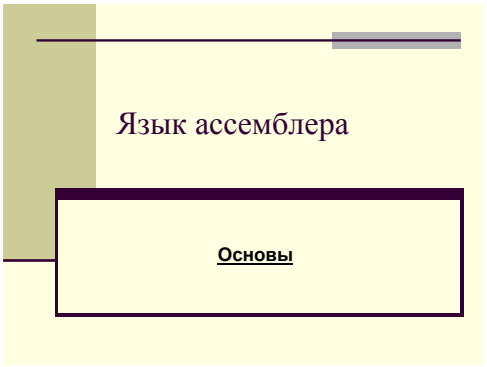
3) Владеть навыками проектирования простых_программных алгоритмов и реализации их на языке программирования.

1. КУРС ЛЕКЦИЙ

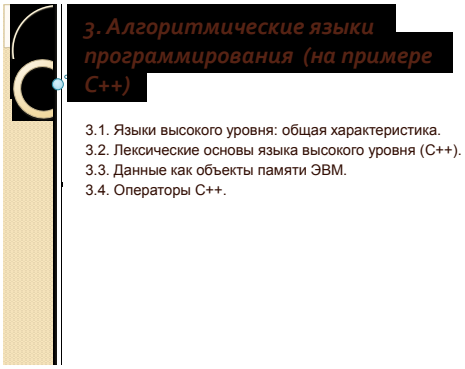
Тема 1. Введение. Микропроцессорная система и ее программирование



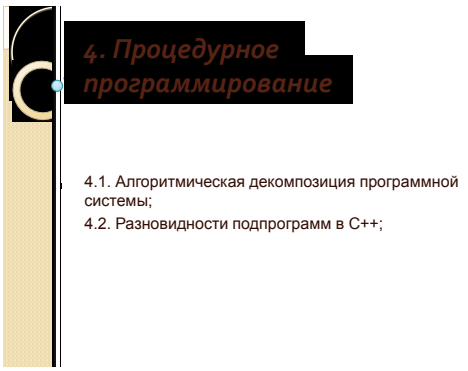
Тема 2. Основы программирования на языке ассемблера



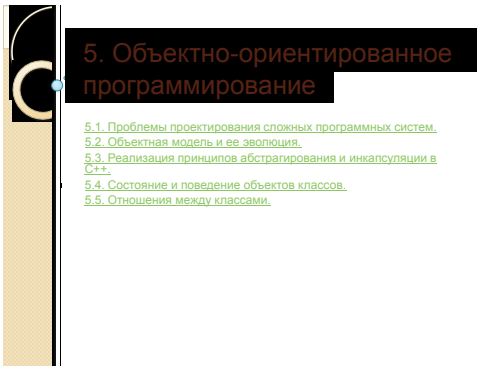
Тема 3. Алгоритмические языки программирования



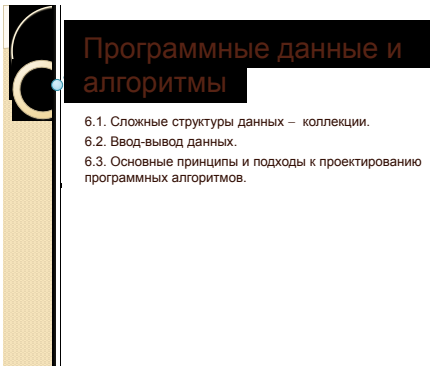
Тема 4. Процедурное программирование



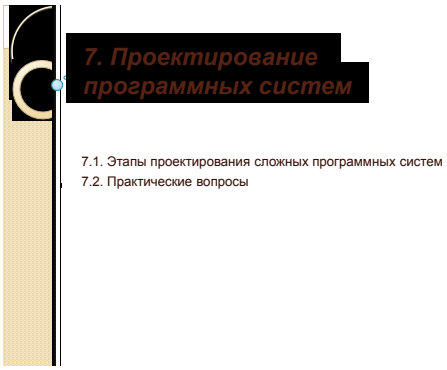
Тема 5. Объектно-ориентированное программирование



Тема 6. Программные данные и алгоритмы



Тема 7. Проектирование программных систем



Тема 8. Программирование графического интерфейса пользователя

8.1. Основы пользовательского интерфейса
8.1.1. Введение
8.1.2. Главное меню
8.1.3. Быстрые кнопки
8.1.4. Компоненты
8.1.5. Дизайнер форм
8.1.6. Редактор кода
8.1.7. Инспектор объектов
8.1.8. Работа с файлами
8.1.9. Контекстное меню
8.1.10. Печать

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К ПРАКТИЧЕСКИМ И ЛАБОРАТОРНЫМ ЗАНЯТИЯМ

Теоретические сведения, задания к работам, методические указания к выполнению и контрольные вопросы приведены в учебных пособиях:

1. Рыбалев, А.Н. Программирование и основы алгоритмизации: пособие к выполнению практических и лабораторных работ/ А. Н. Рыбалев. - Благовещенск: Изд-во Амур. гос. ун-та, 2002 - 92 с.

2. Основы программирования в среде C++ Builder: лаб. практикум по курсу «Основы алгоритмизации и программирования» для студ. 1 – 2-го курсов БГУИР. В 2 ч. Ч. 1 / Бусько В. Л. [и др.] . – Минск: БГУИР, 2007. – 70 с.: ил.

3. Глытина К.У. Информатика: Программирование в Excel и Word на языке VBA. Методические указания по выполнению лабораторных работ для студентов экономических и технических специальностей – Находка: Институт технологии и бизнеса, 2003. – 95 с.

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ КУРСОВОЙ РАБОТЫ

3.1. Этапы выполнения курсовой работы

Курсовая работа по учебной дисциплине «Программирование и основы алгоритмизации» выполняется в соответствии с индивидуальным заданием и предполагает выполнение следующих этапов:

1. Проектирование программы.
2. Согласование результатов проектирования с руководителем курсовой работы.
3. Кодирование и отладка программы.
4. Тестирование программы.
5. Оформление и сдача работы.

3.2. Проектирование программы.

Постановка задачи

Постановка задачи представляет собой точную формулировку задачи и включает в себя:

– содержательную постановку,

– формальную постановку,

– постановку для ЭВМ (спецификацию программы).

Индивидуальные задания на курсовую работу сформулированы в общем виде и требуют уточнений, часть которых может быть определена самим студентом, а часть требует привлечения математического аппарата. Необходимо уяснить задачу, уточнить условия и дать ее формальную постановку в форме математической модели. Следующее действие – определение входных и выходных данных программы в целом.

Постановка задачи является исходным документом для проектирования и кодирования программы. Ошибки и неточности во внешнем проектировании, в конечном счете, трансформируются в ошибки самой программы и обходятся особенно дорого, во-первых, потому, что они делаются на самом раннем этапе разработки программы, и, во-вторых, потому, что они распространяются на последующие этапы. Это требует принятия особенно серьезных мер по их предупреждению. Одна из таких мер – согласование результатов внешнего проектирования с руководителем курсовой работы.

Проектирование интерфейса пользователя

Пользовательский интерфейс представляет средство взаимодействия пользователя с программой. При разработке пользовательского интерфейса следует учитывать потребности, опыт и способности пользователя

В силу большого разнообразия пользователей и видов программных средств существует множество различных стилей пользовательских интерфейсов, при разработке которых могут использоваться разные принципы и подходы. Вопросы проектирования и разработки пользовательского интерфейса подробно рассматриваются во многих последующих учебных дисциплинах. Здесь, назовем только основные принципы, которые следует соблюдать:

пользовательский интерфейс должен базироваться на терминах и понятиях, знакомых пользователю;

пользовательский интерфейс должен быть единообразным;

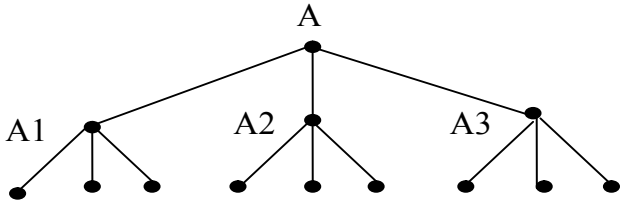
пользовательский интерфейс должен позволять пользователю исправлять собственные ошибки;

пользовательский интерфейс должен предоставлять пользователю справочную информацию.

Определение модульной структуры программы. Метод пошаговой детализации

Алгоритм обычно состоит из нескольких более мелких алгоритмов, каждый из которых в свою очередь можно разложить на составные части и т.д. Алгоритм можно при таком подходе изобразить в виде дерева (рис.1) корнем вверх. Каждый узел дерева обозначает некоторый алгоритм, а отходящие от него вниз линии указывают, из каких частей состоит этот алгоритм. Так, например, составными частями алгоритма А (рис. 1) являются алгоритмы A_1, A_2, A_3 , каждый из которых тоже разделен на составные части.

Разработку алгоритма удобно вести, двигаясь по дереву сверху - вниз, как говорят нисходящим методом, т.е. *от общего к частному.* На первом этапе весь алгоритм представляется в виде одного единственного блока. Затем определяется структура этого блока (одна из трех допустимых структур структурного программирования), т.е. исходный алгоритм разбивается на части. Процесс разработки алгоритма продолжается аналогично, пока весь алгоритм не будет разложен на достаточно простые блоки. Далее, каждый блок разбивается на более мелкие действия до тех пор, пока весь алгоритм не будет разложен на достаточно простые операции.



Таким образом, при проектировании алгоритма решаемая задача поэтапно разбивается на более мелкие и простые задачи, и происходит постепенная детализация и уточнение алгоритма.

При нисходящей разработке программирование модулей программы начинается с модуля самого верхнего уровня (головного), переход к программированию какого-либо другого модуля выполняется в случае, когда если уже запрограммирован модуль, который к нему обращается. После того, как все модули программы запрограммированы, производится их поочередное тестирование и отладка в таком же (нисходящем) порядке. При этом первым тестируется головной модуль программы, который представляет всю тестируемую программу и поэтому тестируется при «естественном» состоянии информационной среды, при котором начинает выполняться эта программа. При этом те модули, к которым может обращаться головной, заменяются их имитаторами (так называемыми заглушкам). Каждый имитатор модуля представляется простым программным фрагментом, который, в основном, сигнализирует о самом факте обращения к имитируемому модулю, производит необходимую для правильной работы программы обработку значений его входных параметров (иногда с их распечаткой) и выдает, если это необходимо, заранее запасенный подходящий результат. После завершения тестирования и отладки головного и любого последующего модуля производится переход к тестированию одного из модулей, которые в данный момент представлены имитаторами, если таковые имеются. Для этого имитатор выбранного для тестирования модуля заменяется самим этим модулем и, кроме того, добавляются имитаторы тех модулей, к которым может обращаться выбранный для тестирования модуль.

Метод восходящей разработки заключается в следующем. Сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с модулей самого нижнего уровня (листья дерева модульной структуры программы), в таком порядке, чтобы для каждого программируемого модуля были уже запрограммированы все модули, к которым он может обращаться. После того, как все модули программы запрограммированы, производится их поочередное тестирование и отладка в принципе в таком же (восходящем) порядке, в каком велось их программирование. Современная технология не рекомендует такой порядок разработки программы, так как каждая программа в какой-то степени подчиняется некоторым внутренним для нее, но глобальным для ее модулей соображениям (принципам реализации, предположениям, структурам данных и т.п.). При восходящей разработке эта глобальная информация для модулей нижних уровней еще не ясна в полном объеме, поэтому часто приходится их перепрограммировать.

Результаты первого этапа работы представляются в виде следующих разделов курсовой работы:

Постановка задачи.

Спецификация программы.

Схема иерархии модулей.

Спецификации модулей.

Проект инструкции пользователя (таблица сообщений).

3.3.Согласование результатов внешнего проектирования с руководителем курсовой работы

Этот этап выполнения работы является необязательным и выполняется по желанию студента.

3.4. Кодирование и отладка программы

При кодировании алгоритма необходимо следовать теории структурного кодирования, которая состоит в получении правильной программы из некоторых простых логических структур. Она базируется на строго доказанной теореме о структурировании, утверждающей, что любую правильную программу (с одним входом и одним выходом, без зацикливаний и недостижимых частей) можно написать с использованием только следующих логических структур:

1) последовательности двух или более операторов;

2) выбора одного из двух операторов;

3) повторения (или управления циклом) оператора, пока выполняется некоторое условие.

Существенно, что каждая из этих конструкций имеет по управлению так же только один вход и один выход. Цель структурного программирования – обеспечить возможность чтения программы от начала до конца, следуя ее логике Фактическое программирование, как и проектирование программы, следует выполнять сверху вниз. При этом главная программа должна быть короткой и вызывать модули, которые можно моделировать, создавая подыгрывающие подпрограммы, или имитаторы. Имитатор – очень короткая последовательность команд, которая используется как замена, пока не будет создана фактическая программа. Имитаторы могут быть двух видов: фиктивные или замещающие модули. Фиктивные модули (иногда называемые «заглушками») не выполняют никакой работы, а только возвращают управление вызывающему модулю. Замещающий модуль выполняет простую обработку до тех пор, пока не окажется возможным программировать более сложный модуль. В простейшем случае замещающий модуль может передавать заранее заготовленный результат.

Большое значение имеет стиль программирования. Под стилем программирования подразумевается набор приемов или методов программирования, которые используют опытные программисты, чтобы получить правильные, эффективные, удобные для применения и легко читаемые программы.

При оформлении текста основной программы и программных модулей целесообразно придерживаться следующих рекомендаций, определяющих практически оправданный стиль программирования:

необходимо использовать комментарии, объясняющие особенности принимаемых решений;

следует использовать осмысленные (мнемонические) и устойчиво различимые имена, не использовать сходные имена и ключевые слова;

необходимо соблюдать осторожность в использовании констант (уникальная константа должна иметь единственное вхождение в текст модуля: при ее объявлении или, в крайнем случае, при инициализации переменной в качестве константы);

дополнительные пробелы (отступы) в начале каждой строки обеспечивают удобочитаемость текста модуля;

Под отладкой понимают деятельность, направленную на обнаружение и исправление ошибок в программе. В том случае, когда программа начинает работать верно, переходят к следующему этапу работы – тестированию. Часто случается так, что после прогона тестов программа вновь должна быть подвергнута отладке.

Таким образом, отладку можно представить в виде многократного повторения трех процессов: тестирования, в результате которого может быть констатировано наличие в программе ошибки, поиска места ошибки в программе и редактирования программы с целью устранения обнаруженной ошибки. Для таких ситуаций, этапы отладки и тестирования программы перекрываются.

3.5. Тестирование программы

Правильность этапов алгоритмизации и программирования на практике обеспечивается путем тестирования, т.е. выполнения алгоритма (программы) со специально подобранными тестами для выявления ошибок. Тест состоит из исходных данных и ожидаемого правильного результат их обработки. Проектирование не слишком большого набора тестов, который обнаружил бы максимальное количество ошибок, является нелегкой творческой задачей.

Исходными данными для разработки тестов являются *спецификация* и *лог*ика *программы*. Спецификация программы — это ее внешнее описание, задающее, **что** должна делать программа. Спецификация содержит *описание входных и выходных данных и функций программы*. Логика программы описывает, **как** программа выполняет свои функции, ее внутреннее устройство (*алгоритм*) и определяется блок-схемой или текстом программы.

Цель тестирования — обнаружение максимального количества из имеющихся в программе ошибок для последующего устранения их. **Наихудшей** из всех методологий проектирования тестов является тестирование со случайными входными величинами.

При разработке тестов для небольших программ или модулей (частей) программы рекомендуется сначала использовать только спецификацию программы (методы ***черного ящика***), а затем проанализировать ее логику для получения дополнительных тестов (методы ***белого ящика*****).** Разработка тестов состоит из двух этапов. ***Черным ящиком*** в кибернетике называется система, рассматриваемая без учета ее внутреннего устройства, только с точки зрения ее внешнего поведения, т.е. зависимости между входными и выходными данными. ***Белый*** *ящик*, в отличие от *черного ящика*,— это система с известным внутренним устройством.

*Методы **черного** ящика*

На основе спецификации программы готовится тест для каждой возможной ситуации (комбинации условий) во входных и выходных данных. Здесь учитывается каждая из допустимых и недопустимых значений входных данных и областей изменения выходных данных. Область может состоять из одного значения. Полезно проверить программу на нежелательные побочные эффекты, например, изменение входных переменных, не являющихся выходными. Для этой проверки можно считать частью ожидаемого результата тестов неизменные входные данные

В программе должны предусматриваться разумные действия при любых, в том числе неправильных или недопустимых входных данных. Программа должна проверять правильность входных данных и при обнаружении ошибок программа обычно сообщает о них, например, печатает соответствующий текст, и, если возможно, исправляет каким-либо образом ошибки и продолжает работу. Худшее, что может сделать программа, – принять неправильные входные данные, а затем выдать неверный, но правдоподобный результат.

Поэтому необходимо построить тест для каждой области недопустимых значений входных данных. Такие «неправильные» тесты зачастую позволяют более эффективно выявлять ошибки в программе, чем «правильные». «Правильный» тест может охватить несколько тестовых ситуаций. «Неправильный» тест должен содержать не более одной ошибки в данных, так как, обнаружив одну ошибку, программа может не найти другие ошибки и правильность реакции программы на них останется непроверенной.

Необходимы также тесты *граничных условий* для границ каждой из областей допустимых значений данных и с незначительным выходом за эти границы. Следует попытаться составить те-

сты, заставляющие программу выйти из области возможных значений выходных данных. При необходимости следует дополнительно разработать тесты, проверяющие случаи, которые по каким-либо предположениям могут быть не учтены при проектировании программы (*метод предположений об ошибке*).

Методы черного ящика полезно использовать до разработки алгоритма на этапе анализа задачи, составления и уточнения спецификации программы. Подготовка тестов на этом этапе способствует более четкой формулировке и глубокому анализу решаемой задачи, служит критерием ее понимания (если разработчик затрудняется в определении результатов работы программы для каких-либо исходных данных, значит, он не до конца понимает задачу и не сможет правильно разработать программу).

Методы белого ящика.

После разработки алгоритма необходимо убедиться, что разработанные тесты обеспечивают выполнение **всех условных переходов** (разветвлений) алгоритма в каждом возможном направлении (*покрытие решений или переходов*). Для сложных условий в циклах и разветвлениях должны тестироваться все возможные комбинации элементарных условий (*комбинаторное покрытие условий*). При этом необходимо выполнение каждого оператора хотя бы один раз (*покрытие операторов*). Тесты должны охватывать основные пути выполнения алгоритма (осуществить покрытие всех путей практически невозможно). Например, для каждого цикла программы должны быть тесты с выполнением тела цикла **ноль, один и максимальное число повторений** (если это возможно). Желательно проверить чувствительность алгоритма к особым значениям входных данных, а также выход за верхние и нижние границы индексов и диапазонов числовых значений. При необходимости следует дополнительно разработать соответствующие тесты.

3.6. Оформление и сдача работы

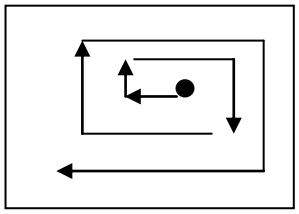
Работа оформляется в форме пояснительной записки и должна содержать следующие разделы, (большая часть разделов была уже названа и пояснена выше):

1. Оглавление.
2. Постановка задачи.
3. Спецификация программы.
4. Схема иерархии модулей.
5. Спецификации модулей.
6. Проект инструкции пользователя (таблица сообщений).
7. Тестовые наборы.
8. Блок-схемы алгоритмов (для каждого модуля).
9. Протоколы тестирования.
10. Листинг программы.

Для сдачи курсовой работы необходима демонстрация работы программы преподавателю, программа демонстрируется как на тестах, предложенных студентом, так и на тестах, которые могут быть предложены руководителем курсовой работы.

3.7. Варианты заданий

№1 Игра «Прыжок кенгуру»

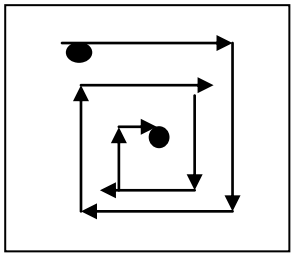


В центре поля 5*5 позиций (можно взять поле другого нечетного размера) стартует кенгуру. Она прыгает по спирали из центра в левый нижний угол либо в соседнюю клетку, либо через одну (случайным образом).

После каждого прыжка кенгуру охотник может поставить ловушку, общее число – не более 3-х. Устанавливая ловушку, охотник

указывает ее координаты (в нижний левый угол ставить ловушку нельзя). Если кенгуру при очередном прыжке попала в ловушку, то она поймана; если благополучно добралась до финиша, “1:0” в пользу общества охраны животных.

№2 Игра «Поймай зайца»

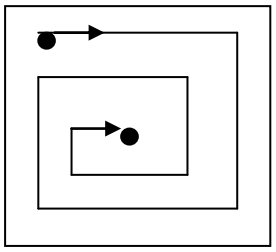


В левом верхнем углу поля 5*5 позиций (можно взять поле другого нечетного размера) стартует заяц. Он прыгает по спирали до центра. Прыгать заяц может либо в соседнюю клетку, либо через одну (случайным образом)

После каждого прыжка охотнику предлагается поставить ловушку. Всего ловушек может быть не более 3-х. Устанавливая ловушку, охотник указывает ее координаты (в центр поля ставить ловушку нельзя).

Если заяц при очередном прыжке попал в ловушку, то он пойман; если благополучно добрался до центра, то охотник остался без зайца.

№3 Игра «Кто вперед»



Два игрока стартуют в левом верхнем углу поля 9*9 и движутся к центру поля. Победит тот, кто придет первым. Положение первого игрока можно отмечать “+”, второго “*”, если оба в одной клетке – “0”.

Игроки могут ходить на одну или две клетки по спирали к центру (игрок вводит, соответственно 1 или 2).

На пути есть 5 особых точек, расставленных программой случайным образом:

Назад на 3 клетки;
Вперед на две клетки;
Пропустить ход;
Сделать дополнительный ход;
Встать в одну клетку с противником
Клетки невидимы до тех пор, пока один из противников не попадет на них.

№4 Игра «Найти невидимку» (диагональная)

На поле n×n в одной из клеток стоит невидимка, которого должен найти игрок. Игрок указывает координаты невидимки. Если угадал, то нашел невидимку. Если не угадал, то невидимка передвигается в другую позицию, а игроку сообщается, где невидимка был в момент хода.

Невидимка может передвигаться только по диагонали.

№5 Игра в кости

Играющий называет любое число в диапазоне от 2 до 12 и ставку, которую он делает в этот ход. Программа с помощью датчика случайных чисел дважды выбирает числа от 1 до 6 (бросает кубик). Если сумма выпавших цифр меньше 7 и играющий задумал число, меньшее 7, он выигрывает сделанную ставку. Если сумма выпавших цифр больше 7 и играющий задумал число больше 7, он также выигрывает сделанную ставку. Если играющий угадал сумму цифр, он получает в четыре раза больше очков, чем сделанная ставка. Ставка проиграна, если не имеет место ни одна из описанных ситуаций.

В начальный момент у играющего 100 очков.

№6 Игра «Коровы и быки» (для слов)

Два игрока загадывают по четырехсимвольному слову (вводят без отображения на экране).

Затем каждый игрок должен угадать слово противника. Игроки ходят по очереди. На каждом шаге игрок называет четырехбуквенное слово, а программа сообщает, сколько букв угадано (быки) и сколько букв угадано и стоит на своем месте(коровы).

№7 Игра «Морской бой» (два игрока)

На поле 4×4 клетки игроки устанавливают 3 корабля по одной клетке (у каждого игрока свое поле). Программа «запоминает» положение кораблей.

Затем игроки начинают поражать корабли противника, по очереди указывая координаты предполагаемого корабля. Результат попадания (попал или мимо) отмечается на поле (например, «*» – попал, «+» – мимо).

№8 Игра «Двадцать одно» (человек - человек)

Два по очереди называют цифры от 1 до 9. Программа суммирует эти цифры.

Проигрывает тот, кто первым дойдет до числа 21 или более.

Например:

Номер хода	Игрок1	Игрок2	Сумма
1	9		9
2		9	18
3	2		20
4		Проиграл	

№9 Игра «Тренировка памяти – числа» (2 игрока)

Играют два игрока. Каждому по очереди на определенное время высвечивается несколько чисел, полученных с помощью датчика случайных чисел. Размер этих чисел ограничен (например, не более 3-х чисел).

Игроки должны воспроизвести числа. Каждому игроку дается определенному число шагов (игроки указывают это число в начале игры). Время запоминания также игроками в начале игры.

Игроки могут играть в 2-х режимах:

А) просто воспроизвести числа;

Б) воспроизвести числа в том же порядке.

№10 Игра «Тренировка памяти – слова» (1 игрок)

Программа на определенное время высвечивает несколько слов- существительных. Слова выбираются и специального файла или из специальной таблицы случайным образом.

Игрок должен воспроизвести слова. Время для запоминания может быть различным. Например, в программе предусматривается три временных режима

Число слов для запоминания может быть различным.

Игрок может играть в 2-х режимах:

А) просто воспроизвести слова;

Б) воспроизвести слова в заданном порядке.

№11 Игра «Жизнь»

Игра моделирует жизнь поколений гипотетической колонии, которые выживают, размножаются или вымирают в соответствии со следующими правилами:

Клетка выживает, если и только если она имеет двух или трех соседей из восьми возможных.

Если у клетки только один сосед ли вовсе ни одного, она погибает в изоляции. Если у клетки четыре или более соседей, она погибает от перенаселения.

В любой пустой позиции, у которой ровно три соседа, в следующем поколении появляется новая клетка.

№12 Игра «Подбери ключи»

Перед играющим четыре запертые двери. Открыть все двери, располагая десятью ключами, каждый из которых может открыть несколько дверей. Предоставляется 14 попыток.

№13 Игра «Ипподром»

Играющий выбирает одну из трех лошадей, состязающихся на бегах, и выигрывает, если его лошадь приходит первой. Скорость лошадей на разных этапах выбирается программой с помощью датчика случайных чисел.

№14 Игра «Угадай слово»

Ведущий вводит слово (без отображения его на экране). На экране высвечивается столько звездочек, сколько букв в слове. Роль ведущего может выполнять программа.

Игрок отгадывает слово по одной букве. Если в слове есть эта буква, она высвечивается вместо звездочки. Игроку всегда сообщается о номере его очередного хода. В любой момент игрок может отказаться от игры.

По окончании игры сообщается, за сколько шагов игрок угадал слово или сколько шагов сделано до прерывания игры, и какое слово было загадано, если игрок отказался от игры.

№15 Игра «Обучение устному счету»

На каждом шаге играющему предлагается два числа и арифметическое действие, которое надо выполнить.

Если игрок отвечает неверно, программа сообщает правильный ответ и дает следующее задание с тем же арифметическим действием.

Размер чисел и максимальное время ответа устанавливаются по желанию игрока в начале игры.

№16 Игра «Крестики – нолики»

№17 Игра «100»

Из кучки, первоначально содержащей 100 спичек, двое играющих поочередно берут по несколько спичек: не менее одной и не более десяти. Проигрывает тот игрок, кто возьмет последнюю спичку.

№18 Игра «НИМ»

Имеется три кучки спичек. Двое играющих по очереди делают ходы. Каждый ход заключается в том, что из одной какой-то кучки берется произвольное ненулевое число спичек. Выигрывает взявший последнюю спичку.

№19 «Цзяньшидзы»

Имеется две кучки камней. Двое играющих по очереди делают ходы. Каждый ход может состоять в одном из двух:

1) Берется произвольное ненулевое число камней и какой-то одной кучки;

2) Берется одновременно по одинаковому ненулевому числу из обеих кучек.

Выигрывает взявший последний камень. Пара (A,B), где A и B - количество камней в кучках при A<B, если число A оканчивается в «фибоначчиевой» системе четным числом нулей, а число B получается из A приписыванием еще одного нуля в конце.

№20 Игра «Семь лунок»

Вдоль доски расположено 7 лунок, в которых лежат 3 черных и 3 белых шара так, как показано на рисунке. Передвинуть черные шары на место белых, а белые - на место черных. Шар можно передвинуть либо в соседнюю с ним пустую лунку, либо в пустую лунку, находящуюся непосредственно за ближайшим шаром.

№21 Игра «Прыгающие шарики».

Эта игра похожа на предыдущую. Исходная позиция - 8 лунок, в которых расставлены 4 черных и 3 белых шара (смотри рис). Поменять местами черные и белые шары. Черные шары можно передвигать только вправо, а белые только влево.

№22 Игра аналог телевизионного шоу «Поле чудес».

№ 23. Программа «Римские цифры»

Программа записи целых чисел в диапазоне от 0 до 3999 римскими цифрами и перевода числа, записанного римскими цифрами в десятичную систему.

Указания: Римские цифры I V X L C D M
(1 5 10 50 100 500 1000)

Правило формирования:

– если большее число стоит перед меньшим, - они складываются (XI = 10+1 = 11);

– если меньшее число стоит перед большим, - от большего отнимается меньшее (IX = 10 - 1 = 9);

– числа M, C, X, I могут «повторяться» (рядом) три раза;

– числа D, L, V не могут повторяться.

Принципы формирования римских цифр рассмотрим на примере:

Числа от 1 до 110:

I II III IV V VI VII VIII IX X (10) XI XII XIII XIV XV XVI XVII XVIII XIX XX (20)

XXI XXII XXIII XXIV XXV XXVI XXVII XXVIII XXIX XXX (30) XXXI XXXII XXXIII XXXIV XXXV XXXVI XXXVII XXXVIII XXXIX XL (40) XLI XLII XLIII XLIV XLV XLVI XLVII XLVIII XLIX L (50) LI LII LIII LIV LV LVI LVII LVIII LIX LX (60)

LXI LXII LXIII LXIV LXV LXVI LXVII LXVIII LXIX LXX (70) LXXI LXXII LXXIII LXXIV LXXV LXXVI LXXVII LXXVIII LXXIX LXXX (80) LXXXI LXXXII LXXXIII LXXXIV LXXXV LXXXVI LXXXVII LXXXVIII LXXXIX XC (90) XCI XCII XCIII XCIV XCV XCVI XCVII XCVIII XCIX C (100) CI CII CIII CIV CV CVI CVII CVIII CIX CX (110)

Максимальное число:

MMMCMXCIX = 3999 = MMM(3000)+CM(900)+XC(90)+IX(9)

№ 24 Программа «Календарь»

Программа, выдающая по запросу пользователя календарь любого года или месяца любого года с указанием дней недели. Если наряду с годом и месяцем введено также и число, программа должна вывести соответствующий день недели. В календаре программа должна выделять выходные и праздничные дни.

Указание: Основной упор сделать на удобство интерфейса пользователя.

№25 Программа «Шифр Гронсфельда»

Программа, шифрующая и дешифрующая обыкновенные текстовые ASCII файлы с помощью усовершенствованного шифра Гронсфельда.

Указание: Шифр Гронсфельда имеет ключ - 5 (в нашем случае любое количество) цифр. Шифруемый текст разбивается на группы символов (пробелы - не исключение) по числу цифр в ключе. Код первого символа группы увеличивается на число, соответствующее первой цифре

ключа, код второго на число, соответствующее второй цифре ключа и т.д. При этом коды меньше 32 (т.н. управляющие символы) преобразованию не подлежат (во избежание повреждения структуры файла и других неприятных последствий). При дешифровке производится обратный процесс (уменьшение кодов).

Если полученный при шифровании код КОД больше 255, применяем формулу КОД = КОД - 224.

Если полученный при дешифрировании код КОД меньше 32 применяем формулу КОД = КОД + 224.

Таким образом, зацикливаем последовательность кодов 32 - 255.

Естественно, при шифровке и дешифровке программа должна запрашивать у пользователя ключ.

№26 Программа «Численное интегрирование»

Программа, выполняющая численное интегрирование системы обыкновенных дифференциальных уравнением любым из методов.

Описание системы диф. уравнений, начальных условий и временного интервала пользователь задает отдельной функцией и глобальными переменными, формат которых (имена, типы, сигнатуры) должен быть задан, так как функция, выполняющая интегрирование, не должна зависеть от конкретной задачи.

Методы численного интегрирования широко описаны в литературе по математике. Выберите наиболее простой (например, метод Эйлера).

№27 Программа «Ход конем»

Программа, находящая решение следующей задачи: необходимо обойти все клетки шахматного поля конем, ни разу не ступив дважды ни на одну клетку.

Задача решается методом проб. Из любой позиции шахматной доски конь может выбрать не более 8 вариантов следующего хода. Программа выбирает произвольный вариант из числа допустимых до тех пор, пока не возникнет ситуация, когда возможных вариантов нет. Тогда делается возврат на один ход назад и выбирается другой вариант хода. Естественно для каждой позиции все "испробованные" варианты должны фиксироваться. Таким образом, конь может вернуться на несколько ходов назад, если при каждом возвращении на один ход оказывается, что для данной позиции все варианты были перебраны.

№28 Программа «База данных»

Программа работы с базой данных фирмы.

База данных должна содержать, как минимум, следующие сведения:

– фамилия, имя, отчество сотрудника, дата (число, месяц, год) рождения;

– занимаемая должность, должностной оклад;

– информация о сотрудниках, находящихся в подчинении, информация о начальнике;

и другую информацию (чем больше, тем лучше).

Программа должна обрабатывать следующие запросы пользователя:

– добавление сотрудника в базу данных и его удаление из нее;

– вывод полной информации о сотруднике по его ФИО;

– вывод информации о всех сотрудниках, родившихся: 1) в определенном месяце, 2) ранее или позднее определенного года;

– информация о сотрудниках, занимающих определенную должность;

– информация о сотрудниках, находящихся в подчинении у определенного сотрудника;

и другие запросы (чем больше, тем лучше).

№29 Программа "Лабиринт"

Разработать, реализовать и протестировать класс Labirint для описания лабиринта и нахождения пути его прохождения. Лабиринт состоит из перекрестков, связывающих 3 или 4 перехода (т.е, попав на перекресток, можно выбрать один из 2 или 3 вариантов дальнейшего маршрута)

Класс должен включать:

Данные:

– массив структур (или указателей на структуры), описывающих каждый перекресток;

– номер точки выхода (индекс элемента массива – «последнего перекрестка»).

Методы:

– конструктор – считывает информацию о лабиринте из файла;

– прохождение лабиринта (возможно, рекурсивный метод) - выводит на экран путь прохождения, т.е. последовательность перекрестков.

В начале пути мы находимся на первом перекрестке.

№30 Программа «Экзаменационный билет»

Экзаменационный билет состоит из двух вопросов (первого и второго). У преподавателя имеется два текстовых файла со списками вопросов (формат которых необходимо продумать).

Программа вызывается каждый раз для очередного студента и должна:

– запросить его фамилию;

– случайным образом выбрать варианты первого и второго вопросов из двух файлов, причем те, что до этого не использовались. В общем случае номера вариантов первого и второго вопросов не должны совпадать, т.е. они выбираются отдельно друг от друга;

– вывести фамилию студента, первый и второй вопросы на экран и принтер;

– сохранить для преподавателя информацию об использованных вопросах в текстовом файле в формате:

Фамилия Номер_первого_вопроса Номер_второго_вопроса

Этот файл должен дополняться после каждого вызова программы завершиться.

№31 Программа «Простейший генетический алгоритм (ПГА)»

Разработать, реализовать и протестировать функцию решения оптимизационной задачи с помощью ПГА.

ПГА применяется для решения задач оптимизации (поиска минимума или максимума) как универсальный численный метод.

Описание метода:

ПГА использует некоторое множество особей – «хромосом» Wi, каждая из которых обладает т.н. функцией пригодности (полезности) f(Wi), характеризующей ее качество. Хромосомы в популяции конкурируют за право участвовать в синтезе новых поколений (потомков). Система стремится максимизировать свою общую (или среднюю) функцию пригодности, двигаясь от поколения к поколению. При этом каждая новая популяция должна накапливать положительные свойства предшествующих. Синтез каждого нового поколения осуществляется при помощи генетических операторов. В ПГА их три: воспроизведение, скрещивание и мутация.

При воспроизведении особи текущего поколения копируются в следующее с вероятностью, пропорциональной функции пригодности особи. Т.о. особи с более высоким значением f(Wi) имеют больший шанс "попасть" в будущее поколение.

Скрещивание состоит в следующем:

1) из вновь воспроизведенных особей случайно выбирается некоторое количество пар;

2) особи пар разрываются по некоторой позиции и обмениваются частями, образуя новые пары. Например, в исходном положении W1 = 1101|0111, W2 = 1001|1100, после скрещивания W1 = 1101|1100, W2 = 1001|0111 Мутация осуществляется путем случайной инверсии значения в некоторой позиции.

Разрабатываемая функция должна в качестве параметров принимать:

1) число особей поколения;

2) число поколений;

3) указатель на функцию, вычисляющую f(Wi). Каждая особь представляется одним байтом.

С помощью функции решить задачу поиска минимума min(f(W)) = min(W-60)^2), где W лежит в пределах 0..255 (unsigned char).

Решение W = 60 (00111100).

№32 Программа «Системы исчисления»

Программа, преобразующая любое (целое или дробное) число из любой системы исчисления в любую другую систему исчисления.

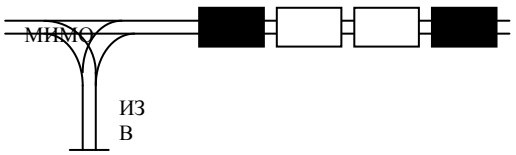
Указание:

При преобразованиях пользуйтесь оператором «деление по модулю».

№33 Программа «Задача восьми ферзей»

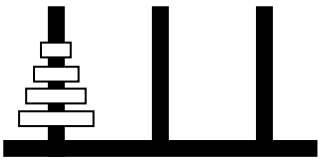
Расположите 8 ферзей на шахматной доске так, что ни один ферзь не убьет другого. Разработать алгоритм с отходами назад, создающий такую конфигурацию. Передвижение и расположение фигур наблюдается во время работы программы.

№34 Программа «Железнодорожный узел»



Железнодорожный сортировочный узел устроен так, как показано на рисунке. На правой стороне собрано в произвольном порядке несколько вагонов обоих типов по N штук. Тупик может вмещать все 2N вагонов. Пользуясь тремя сортировочными операциями: В, ИЗ, МИМО, собрать вагоны на левой стороне так, чтобы типы чередовались. Для решения задачи достаточно 3N-1 сортировочных операций. По запросу пользователя программа должна продемонстрировать правильную сортировку вагонов.

№35 Программа «Ханойская башня»



Доска имеет три колышка. На первом нанизано M дисков убывающего вверх диаметра. Расположить диски в том же порядке на другом колышке. Диски можно перекладывать с колышка на колышек по одному. Класть больший диск на меньший не разрешается. По запросу пользователя программа должна продемонстрировать правильную раскладку дисков.

№36 Программа «Пятнадцать»

На квадратном поле размером 4x4 с помощью датчика случайных чисел расставлены 15 фишек с номерами от 1 до 15. Имеется одна свободная позиция. Расставить фишки по возрастанию их номеров. Передвигать фишки можно только на соседнюю свободную позицию.

№37 Программа дешифрования шифрованного текста с помощью анализа частоты встреч символов

1. Составить простейший шифр в виде соответствий символ – символ (русского алфавита).

2. Зашифровать достаточно большой текст №1 (десятки страниц).

3. Проанализировать достаточно большой текст №2 (десятки страниц) и составить таблицу частоты встреч символов русского языка в виде

«а» – 20383 раз, 13%;

«б» – 114 раз, 3%;

…
Регистр букв учитывать не нужно.
Отсортировать таблицу в порядке убывания частоты.
4. Расшифровать текст №1, используя следующую процедуру:
– составить таблицу встреч символов и отсортировать ее;
– сопоставив таблицы, заменить символы в тексте.

5.МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Самостоятельная работа представляет собой особую, высшую степень учебной деятельности. Она обусловлена индивидуальными психологическими различиями обучающегося и личностными особенностями и требует высокого уровня самосознания, рефлективности. Самостоятельная работа может осуществляться как во внеаудиторное время (дома, в лаборатории), так и на аудиторных занятиях в письменной или устной форме.

Самостоятельная работа обучающихся является составной частью учебной работы и имеет целью закрепление и углубление полученных знаний и навыков, поиск и приобретение новых знаний, в том числе с использованием автоматизированных обучающих систем, а также выполнение учебных заданий, подготовку к предстоящим занятиям, зачетам и экзаменам. Организуется, обеспечивается и контролируется данный вид деятельности студентов соответствующими кафедрами.

Самостоятельная работа предназначена не только для овладения дисциплиной, но и для формирования навыков самостоятельной работы вообще, в учебной, научной, профессиональной деятельности, способности принимать на себя ответственность, самостоятельно решить проблему, находить конструктивные решения, выход из кризисной ситуации и т. д. Значимость самостоятельной работы выходит далеко за рамки отдельного предмета, в связи с чем выпускающие кафедры должны разрабатывать стратегию формирования системы умений и навыков самостоятельной работы. При этом следует исходить из уровня самостоятельности абитуриентов и требований к уровню самостоятельности выпускников, с тем, чтобы за весь период обучения достаточный уровень был достигнут.

При проведении самостоятельной работы, связанной с проработкой теоретического материала, студентам предлагается законспектировать рассматриваемый вопрос, в случае необходимости задать возникшие вопросы на практическом занятии или на консультации.

При изучении дисциплины практикуются следующие виды и формы самостоятельной работы студентов:

- подготовка к устному опросу по темам лабораторных работ;
- подготовка к тестированию;
- выполнение разделов курсовой работы.

Самостоятельная работа тесно связана с контролем (контроль также рассматривается как завершающий этап выполнения самостоятельной работы), при выборе вида и формы самостоятельной работы следует учитывать форму контроля.

Формы контроля при изучении дисциплины:

- устный опрос;
- тестирование;
- проверка и защита курсовой работы.

Самостоятельная работа проводится в виде подготовительных упражнений для усвоения нового, упражнений при изучении нового материала, упражнений в процессе закрепления и повторения, упражнений проверочных и контрольных работ, а также для самоконтроля.

Для организации самостоятельной работы необходимы следующие условия:

- готовность студентов к самостоятельному труду;
- наличие и доступность необходимого учебно-методического и справочного материала;
- консультационная помощь.

Самостоятельная работа может проходить в лекционном кабинете, лаборатории, компьютерном зале, библиотеке, дома. Самостоятельная работа тренирует волю, воспитывает работоспособность, внимание, дисциплину и т.д.

Рекомендации по организации аудиторной самостоятельной работой

Аудиторная самостоятельная работа по дисциплине выполняется на учебных занятиях под непосредственным руководством преподавателя и по его заданию.

Основными видами аудиторной самостоятельной работы являются:

выполнение лабораторных и практических работ по инструкциям;

работа с литературой и другими источниками информации, в том числе электронными;

само- и взаимопроверка выполненных заданий;

Выполнение практических и лабораторных работ осуществляется на практических и лабораторных занятиях в соответствии с графиком учебного процесса. Работа с литературой, другими источниками информации, в т.ч. электронными может реализовываться на лекционных и практических занятиях. Данные источники информации могут быть представлены на бумажном и/или электронном носителях, в том числе, в сети Internet. Преподаватель формулирует цель работы с данным источником информации, определяет время на проработку документа и форму отчетности.

Само- и взаимопроверка выполненных заданий чаще используется на лекционном, практическом занятии и имеет своей целью приобретение таких навыков как наблюдение, анализ ответов сокурсников, сверка собственных результатов с эталонами.

Рекомендации по организации внеаудиторной самостоятельной работы

Внеаудиторная самостоятельная работа выполняется по заданию преподавателя, но без его непосредственного участия.

При предъявлении видов заданий на внеаудиторную самостоятельную работу рекомендуется использовать дифференцированный подход к уровню подготовленности обучающегося. Перед выполнением внеаудиторной самостоятельной работы преподаватель проводит консультацию с определением цели задания, его содержания, сроков выполнения, ориентировочного объема работы, основных требований к результатам работы, критериев оценки, форм контроля и перечня литературы. В процессе консультации преподаватель предупреждает о возможных типичных ошибках, встречающихся при выполнении задания.

Самостоятельная работа может осуществляться индивидуально или группами студентов в зависимости от цели, объема, конкретной тематики самостоятельной работы, уровня сложности, уровня подготовленности обучающихся.

Видами заданий для внеаудиторной самостоятельной работы могут быть:

для овладения знаниями: чтение текста (учебника, первоисточника, дополнительной литературы); составление плана текста; графическое изображение структуры текста; конспектирование текста; выписки из текста; работа со словарями и справочниками; учебно-исследовательская работа; использование аудио- и видеозаписей, компьютерной техники и Интернет-ресурсов и др.;

для закрепления и систематизации знаний: работа с конспектом лекции (обработка текста); повторная работа над учебным материалом (учебника, первоисточника, дополнительной литературы, аудио- и видеозаписей); составление плана и тезисов ответа; составление таблиц, глоссария для систематизации учебного материала; изучение словарей, справочников; ответы на контрольные вопросы; аналитическая обработка текста (аннотирование, рецензирование, реферирование, контент-анализ и др.); подготовка сообщений к выступлению на семинаре, конференции; подготовка рефератов, докладов; составление библиографии, заданий в тестовой форме и др.;

для формирования умений: решение задач и упражнений по образцу; решение вариативных задач и упражнений; составление схем; проектирование и моделирование разных видов и компонентов профессиональной деятельности и др.

Для обеспечения внеаудиторной самостоятельной работы по дисциплине преподавателем разрабатывается перечень заданий для самостоятельной работы, который необходим для эффективного управления данным видом учебной деятельности обучающихся.

Преподаватель осуществляет управление самостоятельной работой, регулирует ее объем на одно учебное занятие и осуществляет контроль выполнения всеми обучающимися группы. Для удобства преподаватель может вести ведомость учета выполнения самостоятельной работы, что позволяет отслеживать выполнение минимума заданий, необходимых для допуска к итоговой аттестации по дисциплине.

В процессе самостоятельной работы студент приобретает навыки самоорганизации, самоконтроля, самоуправления и становится активным самостоятельным субъектом учебной деятельности.

Обучающийся самостоятельно определяет режим своей внеаудиторной работы и меру труда, затрачиваемого на овладение знаниями и умениями по каждой дисциплине, выполняет внеаудиторную работу по индивидуальному плану, в зависимости от собственной подготовки, бюджета времени и других условий.

Ежедневно обучающийся должен уделять выполнению внеаудиторной самостоятельной работы в среднем не менее 3 часов.

При выполнении внеаудиторной самостоятельной работы обучающийся имеет право обращаться к преподавателю за консультацией с целью уточнения задания, формы контроля выполненного задания.

Контроль результатов внеаудиторной самостоятельной работы студентов может проводиться в письменной, устной или смешанной форме с представлением продукта деятельности обучающегося. В качестве форм и методов контроля внеаудиторной самостоятельной работы могут быть использованы зачеты, тестирование, самоотчеты, контрольные работы, защита творческих работ и др.

Методические рекомендации по изучению теоретических основ дисциплин

Изучение теоретической части дисциплин призвано не только углубить и закрепить знания, полученные на аудиторных занятиях, но и способствовать развитию у студентов творческих навыков, инициативы и организовать свое время.

Самостоятельная работа при изучении дисциплин включает:

- чтение студентами рекомендованной литературы и усвоение теоретического материала дисциплины;
- знакомство с Интернет-источниками;
- подготовку к различным формам контроля (тесты, опрос по темам лабораторных работ);
- подготовку ответов на вопросы по различным темам дисциплины в той последовательности, в какой они представлены.

Планирование времени, необходимого на изучение дисциплин, студентам лучше всего осуществлять весь семестр, предусматривая при этом регулярное повторение материала.

Материал, законспектированный на лекциях, необходимо регулярно прорабатывать и дополнять сведениями из других источников литературы, представленных не только в программе дисциплины, но и в периодических изданиях.

При изучении дисциплины сначала необходимо по каждой теме прочитать рекомендованную литературу и составить краткий конспект основных положений, терминов, сведений, требующих запоминания и являющихся основополагающими в этой теме для освоения последующих тем курса. Для расширения знания по дисциплине рекомендуется использовать Интернет-ресурсы; проводить поиски в различных системах и использовать материалы сайтов, рекомендованных преподавателем.

При ответе на экзамене и зачете необходимо: продумать и четко изложить материал; дать определение основных понятий; дать краткое описание явлений; привести примеры. Ответ следует иллюстрировать схемами, рисунками и графиками.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Львович И.Я. Основы информатики [Электронный ресурс]: учебное пособие/ Львович И.Я., Преображенский Ю.П., Ермолова В.В.— Электрон. текстовые данные.— Воронеж: Воронежский институт высоких технологий, 2014.— 339 с.— Режим доступа: http://www.iprbookshop.ru/23359.— ЭБС «IPRbooks», по паролю

2. Гарибов А.И. Информатика [Электронный ресурс]: учебное пособие/ Гарибов А.И., Куценко Д.А., Бондаренко Т.В.— Электрон. текстовые данные.— Белгород: Белгородский государственный технологический университет им. В.Г. Шухова, ЭБС АСВ, 2012.— 224 с.— Режим доступа: http://www.iprbookshop.ru/27282.— ЭБС «IPRbooks», по паролю

3. Кирнос В.Н. Информатика 2. Основы алгоритмизации и программирования на языке С++ [Электронный ресурс]: учебно-методическое пособие/ Кирнос В.Н.— Электрон. текстовые данные.— Томск: Томский государственный университет систем управления и радиоэлектроники, Эль Контент, 2013.— 160 с.— Режим доступа: http://www.iprbookshop.ru/14011.— ЭБС «IPRbooks», по паролю

4.Губарев В.В. Введение в теоретическую информатику. Часть 1 [Электронный ресурс]: учебное пособие/ Губарев В.В.— Электрон. текстовые данные.— Новосибирск: Новосибирский государственный технический университет, 2014.— 420 с.— Режим доступа: http://www.iprbookshop.ru/44907.— ЭБС «IPRbooks», по паролю

5.Прохорова О.В. Информатика [Электронный ресурс]: учебник/ Прохорова О.В.— Электрон. текстовые данные.— Самара: Самарский государственный архитектурно-строительный университет, ЭБС АСВ, 2013.— 106 с.— Режим доступа: http://www.iprbookshop.ru/20465.— ЭБС «IPRbooks», по паролю

6.Забуга А.А. Теоретические основы информатики [Электронный ресурс]: учебное пособие/ Забуга А.А.— Электрон. текстовые данные.— Новосибирск: Новосибирский государственный технический университет, 2013.— 168 с.— Режим доступа: http://www.iprbookshop.ru/45037.— ЭБС «IPRbooks», по паролю

7.Гураков А.В. Информатика. Введение в Microsoft Office [Электронный ресурс]: учебное пособие/ Гураков А.В., Лазичев А.А.— Электрон. текстовые данные.— Томск: Томский государственный университет систем управления и радиоэлектроники, Эль Контент, 2012.— 120 с.— Режим доступа: http://www.iprbookshop.ru/13934.— ЭБС «IPRbooks», по паролю

8.Устинов В.В. Основы алгоритмизации и программирования. Часть 1 [Электронный ресурс]: конспект лекций/ Устинов В.В.— Электрон. текстовые данные.— Новосибирск: Новосибирский государственный технический университет, 2010.— 40 с.— Режим доступа: http://www.iprbookshop.ru/44676.— ЭБС «IPRbooks», по паролю

9.Устинов В.В. Основы алгоритмизации и программирование. Часть 2 [Электронный ресурс]: конспект лекций/ Устинов В.В.— Электрон. текстовые данные.— Новосибирск: Новосибирский государственный технический университет, 2013.— 32 с.— Режим доступа: http://www.iprbookshop.ru/44675.— ЭБС «IPRbooks», по паролю

10.Рыбалев, А.Н. Программирование и основы алгоритмизации: Лабораторный практикум / А. Н. Рыбалев ; АмГУ, Эн.ф. - Благовещенск : Изд-во Амур. гос. ун-та, 2002. - 91 с. http://irbis.amursu.ru/DigitalLibrary/AmurSU_Edition/122.pdf

11.Галаган, Т. А. Язык С++. Сборник задач и тестовых заданий [Электронный ресурс] : учеб. пособие / Т. А. Галаган ; АмГУ, ФМиИ. - Благовещенск : Изд-во Амур. гос. ун-та, 2014. - 112 с. http://irbis.amursu.ru/DigitalLibrary/AmurSU_Edition/6747.pdf

12.Галаган, Т. А. Язык С++. Сборник задач и тестовых заданий [Электронный ресурс] : учеб. пособие / Т. А. Галаган ; АмГУ, ФМиИ. - Благовещенск : Изд-во Амур. гос. ун-та, 2014. - 112 с. http://irbis.amursu.ru/DigitalLibrary/AmurSU_Edition/6747.pdf