Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Амурский государственный университет»

Кафедра автоматизации производственных процессов и электротехники

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ДИСЦИПЛИНЫ

Программирование и основы алгоритмизации

Основной образовательной программы по направлению подготовки (специальности)

220301 Автоматизация технологических процессов и производств (по отраслям)

Благовещенск 2012

УМКД разработан канд. техн. наук, доцентом Рыбалевым Андреем Николаевичем

Рассмотрен и рекомендован на заседании кафедры

Протокол заседания кафедры от «23» января 2012 г. № 5

Зав. кафедрой _____ / А.Н. Рыбалев

УТВЕРЖДЕН

Протокол заседания УМСС Автоматизация технологических процессов и электротехники

от «23» января 2012 г. № 5

Председатель УМСС _____ / А.Н. Рыбалев

1. РАБОЧАЯ ПРОГРАММА

1.1. ЦЕЛИ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Целью данного курса 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.

1.2. МЕСТО ДИСЦИПЛИНЫ В СТРУКТУРЕ ООП ВПО

Дисциплина базируется на курсах «Математика» (численные методы решения задач) и «Информатика» (алгоритмы и программирование). Знания и умения, приобретенные студентами при изучении дисциплины, найдут применение при освоении всех дисциплин, требующих составления алгоритмов и программ для решения различных задач с помощью вычислительной техники, а также при курсовом и дипломном проектировании и в практической деятельности выпускника.

1.3. КОМПЕТЕНЦИИ ОБУЧАЮЩЕГОСЯ, ФОРМИРУЕМЫЕ В РЕЗУЛЬТАТЕ ОСВОЕНИЯ ДИСЦИПЛИНЫ (МОДУЛЯ)

В результате освоения дисциплины обучающийся должен демонстрировать следующие результаты образования:

1) Знать:

структуру и состав микропроцессорной системы, назначение и состав системного и прикладного программного обеспечения;

основы, возможности типовые конструкции языка ассемблера;

лексические основы, типы данных и основные операторы языка высокого уровня;

способы декомпозиции программной системы, основные принципы объектно-ориентированного программирования и их реализацию в языке высокого уровня;

основные структуры коллекций данных и алгоритмы работы с ними;

этапы проектирования программных систем.

2) Уметь:

составлять алгоритмы решения задач обработки данных;

реализовывать алгоритмы на языке высокого уровня;

проектировать программные системы в рамках как процедурного, так объектно-ориентированного подхода;

разрабатывать программы небольшого уровня сложности.

3) Владеть навыками работы в интегрированных средах разработки программного обеспечения (в т.ч. редактирования, компиляции, отладки программ).

1.4.СТРУКТУРА И СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

Общая трудоемкость дисциплины составляет 120 часов.

№ п/п	Раздел дисциплины	Семестр	Неделя семестра	Виды учебной работы, включая самостоятельную работу студентов и трудоемкость (в часах)				Формы текущего контроля успеваемости *(по неделям семестра)* Форма промежуточной аттестации *(по семестрам)*
				Лекц.	Пр.	Лаб.	СР	
1	Введение. Микропроцессорная система и ее программирование	4	1	2				Контрольная точка и тестирование №1, экзамен
2	Основы программирования на языке ассемблера	4	2,3	4				Контрольная точка и тестирование №1, экзамен
3	Алгоритмические языки программирования	4	4,5,6	6	5	5	8	Контрольная точка и тестирование №1, контроль выполнения практических и лабораторных работ, курсовая работа, экзамен
4	Процедурное программирование	4	7,8,	4	3	3	10	Контрольная точка и тестирование №2, контроль выполнения практических и лабораторных работ, курсовая работа, экзамен
5	Объектно-ориентированное программирование	4	9,10, 11	6	5	5	10	Контрольная точка и тестирование №2, контроль выполнения практических и лабораторных работ, курсовая работа, экзамен
6	Программные данные и алгоритмы	4	12,13, 14, 15,16	10	5	5	10	Контроль выполнения практических и лабораторных работ, курсовая работа, экзамен
7	Проектирование программных систем	4	17,18	4			10	Контроль выполнения практических и лабораторных работ, курсовая работа, экзамен

1.5. СОДЕРЖАНИЕ РАЗДЕЛОВ И ТЕМ ДИСЦИПЛИНЫ

1.5.1. Лекции

1. Введение. Микропроцессорная система и ее программирование.

Состав микропроцессорной системы. Микропроцессоры, память, периферийные устройства, общие понятия. Система команд микропроцессора. Классификация команд. Адресация памяти. Сегментация памяти.

2. Основы программирования на языке ассемблера.

Основы ассемблера. Директивы определения данных. Команды. Директивы подготовки сегментов. Оформления подпрограмм. Общая структура программы на языке ассемблера. Макросредства ассемблера.

3. Алгоритмические языки программирования.

Языки высокого уровня общая характеристика. История развития языков высокого уровня.

Разработка программных продуктов с помощью языков высокого уровня. Этапы разработки. Препроцессорная обработка. Трансляция и интерпретация. Компиляция и компоновка.

Лексические основы языка высокого уровня. Идентификаторы и служебные слова. Константы. Знаки операций, разделители.

Данные как объекты памяти ЭВМ. Понятие типа данных. Типизация. Основные и производные типы данных. Определения и описание данных. Атрибуты типов памяти: класс памяти, видимость, продолжительность существования, тип компоновки. Статические и динамические данные. Агрегатные типы данных. Массивы. Структурированные типы данных.

Операторы. Операторы выбора, цикла, передачи управления. Процедуры и функции.

4. Процедурное программирование.

Алгоритмическая декомпозиция программной системы. Основные виды подпрограмм: процедуры и функции. Механизмы передачи параметров и возврата результатов. Определение, описание и вызов функций и процедур.

Разновидности подпрограмм: с переменным числом параметров, рекурсивные, подставляемые.

Структура программы процедурного типа. Принцип модульности. Разделение подпрограмм по модулям в сложных системах.

5. Объектно-ориентированное программирование.

Проблемы проектирования сложных программных систем. Объектно-ориентированная декомпозиция.

Объектная модель и ее эволюция. Объектно-ориентированные анализ, проектирование и программирование. Составные части объектного подхода. Абстрагирование. Инкапсуляция. Модульность. Иерархия. Типизация. Параллелизм.

Классы и объекты. Состояние и поведение объекта. Идентичность объекта. Отношения между объектами. Классы. Интерфейс и реализация. Жизненный цикл. Ассоциация, наследование и агрегация классов. Отношения использования. Шаблоны классов.

6. Программные данные и алгоритмы.

Сложные структуры данных: списки деревья, сети. Ввод-вывод данных. Потоки ввода-вывода. Файлы. Типы файлов.

Основные принципы и подходы к проектированию программных алгоритмов. Классы алгоритмов. Методы частных целей, подъема ветвей и границ, эвристический метод. Итерационные и рекурсивные алгоритмы, особенности их программной реализации, преимущества и недостатки. Методы сортировки и поиска данных. Методы линейного и бинарного поиска. Методы сортировки: вставками, пузырьковая, быстрая сортировки.

7. Проектирование программных систем.

Этапы проектирование сложных программных систем. Инструменты проектирования. Критерии оценки проекта. Микро- и макропроцессы проектирования. Анализ и эволюция проекта. Стандарты на разработку прикладных программных средств. Практические вопросы: распределение ресурсов, роли разработчиков, интеграция процесса, управление конфигурацией и версиями, тестирование. Повторное использование и качество программного продукта. Документация проекта. Сопровождение и эксплуатация программных средств.

1.5.2. Практические занятия

Практические занятия включают анализ, построение и программирование алгоритмов решения наиболее сложных заданий к лабораторным работам по темам:

1. Базовые типы данных и структура программы на С++.
2. Классы С++.
3. Стеки и очереди.
4. Перегрузка стандартных операций и динамические классы.
5. Связные списки.
6. Деревья.
7. Имитационное моделирование.

1.5.4. Лабораторные занятия
Тематика лабораторных работ:
1. Знакомство со средствами разработки программ. Изучение стадий редактирования, отладки, компиляции. Создание простой одномодульной программы. Компоновка многомодульной программы с подключением библиотек. Создание программного проекта.
2. Базовые типы данных и структура программы на С++.
3. Классы С++.
4. Стеки и очереди.
5. Перегрузка стандартных операций и динамические классы.
6. Связные списки.
7. Деревья.
8. Имитационное моделирование.

1.6. САМОСТОЯТЕЛЬНАЯ РАБОТА

№ п/п	№ раздела (темы) дисциплины	Форма (вид) самостоятельной работы	Трудоёмкость в часах
1	Введение. Микропроцессорная система и ее программирование		
2	Основы программирования на языке ассемблера		
3	Алгоритмические языки программирования	Выполнение КР Подготовка к практическим и лабораторным занятиям, подготовка отчетов	5 3
4	Процедурное программирование	Выполнение КР Подготовка к практическим и лабораторным занятиям, подготовка отчетов	5 5
5	Объектно-ориентированное программирование	Выполнение КР Подготовка к практическим и лабораторным занятиям, подготовка отчетов	5 5
6	Программные данные и алгоритмы	Выполнение КР Подготовка к практическим и лабораторным занятиям, подготовка отчетов	5 5
7	Проектирование программных систем	Выполнение КР	10

1.7.ОБРАЗОВАТЕЛЬНЫЕ ТЕХНОЛОГИИ
1.7.1. Активные инновационные методы обучения: нет.
1.7.2. Технологии обучения: традиционные.

1.7.3. Информационные технологии: мультимедийное обучение (демонстрации на видеопроекторе на лекционных занятиях).

1.7.4. Информационные системы: электронная база учебно-методических ресурсов на основе сайта app.vrsoft.ru.

1.7.5. Инновационные методы контроля: компьютерное тестирование в ходе изучения дисциплины и по ее окончанию.

1.8. ОЦЕНОЧНЫЕ СРЕДСТВА ДЛЯ ТЕКУЩЕГО КОНТРОЛЯ УСПЕВАЕМОСТИ, ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ ПО ИТОГАМ ОСВОЕНИЯ ДИСЦИПЛИНЫ И УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

1.8.1. Вопросы для тестирования, охватывающие основные темы, изучаемые студентами в данном курсе, и сгруппированые по разделам:

1. Состав вычислительной системы. Построение программ.

2. Базовые понятия языка программирования C++.

3. Объектно-ориентированное программирование.

4. Структуры данных и алгоритмы.

Тестирование является составной частью процедуры промежуточного контроля знаний (в ходе изучения дисциплины), а также используется для контроля остаточных знаний (после окончания изучения дисциплины).

1.8.2.Вопросы к экзамену

1) Микропроцессорная система. Состав и характеристики. CISC и RISC процессоры. Микропроцессоры Intel x86.
2) Память микропроцессорной системы. Виды памяти. Применение памяти различных видов в персональном компьютере.
3) Устройства ввода-вывода. Контроллеры устройств. Системные ресурсы, закрепленные за устройствами. Способы взаимодействия микропроцессора с устройствами ввода вывода.
4) Микропроцессоры: структура регистров. Регистры общего назначения, специальные регистры, сегментные регистры. Сегментация памяти.
5) Система команд микропроцессора. Классификация команд. Механизм прерываний.
6) Системное программное обеспечение. Операционная система. Состав и назначение.
7) Основы языка ассемблера. Директивы определения данных. Организация сегментов.
8) Команды языка ассемблера. Общий вид. Примеры. Использование ссылок в качестве параметров.
9) Язык ассемблера: команды организации переходов и циклов.
10) Язык ассемблера: подпрограммы. Механизмы передачи параметров и возврата результата.
11) Построение программ. Интерпретация, компиляция, ассемблирование. Основные этапы построения программы.
12) Языки высокого уровня. Алфавит языка C++. Идентификаторы, константы, знаки операций, разделители.
13) Атрибуты объектов памяти: класс памяти, область действия, видимость, продолжительность существования. Типы данных. Определение и описание объектов памяти. Преобразование типов.
14) Скалярные типы данных (целочисленные, вещественные, символьные и указатели). Указатели, массивы и строки.
15) Пользовательские типы данных. Структуры, классы.
16) Операторы C/C++. Условный оператор, переключатели, операторы цикла. Операторы передачи управления.

17) Подпрограммы. Передача параметров по ссылке и по значению. Реализация подпрограмм в C/C++. Функции. Определение и описание.

18) Функции. Параметры по умолчанию. Функции с переменным числом параметров. Рекурсивные функции. Inline - функции. Перегрузка функций.

19) Объектно-ориентированное программирование. Объектная модель. Объектно-ориентированная декомпозиция. Абстрагирование. Инкапсуляция. Модульность, иерархия.

20) Реализация принципов абстрагирования и инкапсуляции в C++. Классы и их компоненты. Статус доступа к компонентам класса. Статические компоненты. (На примере модели обслуживания).

21) Компонентные функции классов. Определение и описание. Указатель this. Классификация компонентных функций. Конструкторы и деструкторы. Дружественные функции и классы. (На примере модели обслуживания).

22) Отношения между классами (ассоциация, агрегация, наследование). Наследование. Определение производных классов. Доступ к унаследованным компонентам из производных классов. Особенности конструкторов и деструкторов при наследовании. (На примере модели обслуживания).

23) Множественное наследование. Виртуальные базовые классы. Виртуальные функции. Полиморфизм. Абстрактные классы. (На примере модели обслуживания).

24) Механизм шаблонов в C++. Шаблоны функций и классов. (На примере класса Node)

25) Коллекции данных. Организация. Классификация. Доступ к данным.

26) Алгоритмы работы с коллекциями данных. Критерии эффективности.

27) Алгоритмы поиска данных. Последовательный и бинарный поиск и их вычислительная эффективность.

28) Алгоритмы сортировки порядка n^2. Обменная сортировка и сортировка выбором. Анализ вычислительной эффективности.

29) Алгоритмы сортировки порядка n^2. Пузырьковая сортировка и сортировка вставками. Анализ вычислительной эффективности.

30) Алгоритмы сортировки порядка $n*\log_2 n$. Турнирная сортировка. Анализ вычислительной эффективности.

31) Алгоритмы сортировки порядка $n*\log_2 n$. «Быстрая» сортировка.

32) Методы проектирования алгоритмов (подъема ветвей и границ, частных целей, приближенные и эвристические методы).

33) Стеки и очереди. Класс Stack (стек). Данные и методы класса.

34) Стеки и очереди. Класс Queue (очередь). Данные и методы класса.

35) Стеки и очереди. Класс PQueue (очередь приоритетов). Данные и методы класса.

36) Классы и динамическая память. Конструкторы динамических классов. Конструктор копирования. Перегруженный оператор присваивания. Деструктор. (На примере класса Matrix).

37) Перегрузка стандартных операций для объектов классов. (На примере класса Matrix).

38) Связные списки. Структура связного списка. Класс Node. Данные и методы класса.

39) Связные списки. Структура двусвязного списка. Класс DNode. Данные и методы класса.

40) Деревья. Бинарные деревья. Класс TreeNode. Данные и методы класса.

41) Рекурсивные методы прохождения бинарных деревьев.

42) Бинарные деревья поиска. Класс BinSTree. Данные и методы класса.

43) Бинарные деревья поиска. Класс BinSTree. Поиск данных в бинарном дереве поиска. Методы FindNode, Find.

44) Бинарные деревья поиска. Класс BinSTree. Вставка данных в бинарное дерево поиска. Метод Insert.

45) Бинарные деревья поиска. Класс BinSTree. Удаление данных из бинарного дерева поиска. Метод Delete.

46) Модели обслуживания. Подходы к построению. Реализация событийного подхода (классы EventsServers и EventsGenerators). Виды событий в системе обслуживания.

47) Модель обслуживания. Система классов.

48) Модель обслуживания. Механизм взаимодействия основных функций модели.

49) Этапы проектирование сложных программных систем. Микро и макропроцессы проектирования. Анализ и эволюция проекта.

50) Стандарты на разработку прикладных программных средств. Практические вопросы проектирования.

51) Документация проекта. Сопровождение и эксплуатация программных средств.

1.8.3. Учебные пособия для подготовки и выполнения практических и лабораторных работ:

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.

1.9. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ (МОДУЛЯ)

а) основная литература:

1. Непейвода, Н.Н. Стили и методы программирования: курс лекций: учеб. пособие / Н. Н. Непейвода. - М. : Интернет- Ун-т Информ. Технологий, 2005. - 317 с.

2. Давыдов, В.Г. Программирование и основы алгоритмизации: учеб. пособие: рек. УМО/ В. Г. Давыдов. - 2-е изд., стер. - М. : Высш. шк., 2005. - 448 с.

3. Гвоздева, В.А. Введение в специальность программиста: учеб.: рек. Мин. обр. РФ / В. А. Гвоздева. - М. : ФОРУМ : ИНФРА - М, 2005. - 207 с.

б) дополнительная литература:

1. Валюк, Т.В. Программирование на СИ/СИ++: учеб. - метод. пособие / Т. В. Валюк, Л. А. Соловцова ; АмГУ, ФМиИ. - Благовещенск : Изд-во Амур. гос. ун-та, 1999. - 62 с.

2. 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.

3. Макконелл, Дж. Основы современных алгоритмов: учеб. пособие / Дж. Макконелл ; пер. с англ., под ред. С. К. Ландо. - 2-е изд., доп. - М. : Техносфера, 2006. - 368 с.

4. Шпехт, И.А. Информатика: алгоритмизация и основы программирования: учеб. пособие / И. А. Шпехт, Р. Р. Саакян ; АмГУ, Фак. мат. и информ. - Благовещенск : Изд-во Амур. гос. ун-та, 2005. - 80 с.

5. Окулов, С.М. Программирование в алгоритмах: учеб. пособие / С. М. Окулов . - М.: БИНОМ. Лаб. знаний, 2004. - 342 с.

6. Галаган, Т.А. Язык программирования С ++ в примерах и задачах: практикум по дисциплине «Алгоритмические языки и программирование» / Т. А. Галаган, Л. А. Соловцова; АмГУ, ФМиИ. - Благовещенск : Изд-во Амур. гос. ун-та, 2005. - 104 с.

в) программное обеспечение и Интернет-ресурсы

Программное обеспечение:

1) ОС Microsoft Windows 2000, Microsoft Windows XP;

2) MS Office (Word, PowerPoint);

3) Интегрированные среды разработки Borland C++, Borland C++ Builder.

Интернет-ресурсы:

№	Наименование ресурса	Краткая характеристика
1	http://www.edu.ru/	Российское образование. Феде-

		ральный портал
2	http://app.vrsoft.ru	Сайт кафедры АПП и Э, библиотека электронных ресурсов
3	http://www.codenet.ru/	Сайт, посвященный вопросам программирования. Библиотека ресурсов
4	http://www.helloworld.ru/	Документация и книги по программированию

1. 10.МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ (МОДУЛЯ)

1) Видеопроектор;

2) Компьютерный класс (ауд. 402 корп. №6) для выполнения лабораторных работ.

1.11. РЕЙТИНГОВАЯ ОЦЕНКА ЗНАНИЙ СТУДЕНТОВ ПО ДИСЦИПЛИНЕ

не предусмотрена

2. КРАТКОЕ ИЗЛОЖЕНИЕ ПРОГРАММНОГО МАТЕРИАЛА

2.1. Введение. Микропроцессорная система и ее программирование

2.1.1. План лекции

Состав микропроцессорной системы.

Микропроцессоры: определение. Однокристальные и секционные микропроцессоры, общего назначения и специализированные. CISC и RISC процессоры. Микропроцессоры семейства Intel 86.

Память микропроцессорной системы. Виды памяти. Оперативная память. Статическая память с произвольной выборкой. Динамическая память с произвольной выборкой. ПЗУ и ППЗУ. Применение памяти различных видов в персональном компьютере.

Устройства ввода-вывода. Стандартные устройства. Контроллеры устройств. Обращение к устройствам: выделение адресного пространства, номеров прерываний и каналов DMA.

Связь компонентов микропроцессорной системы. Система внутренних шин. Организация обмена данными.

Программирование микропроцессорной системы.

Структура микропроцессора: система регистров. Регистры общего назначения, регистр флагов, сегментные регистры.

Система команд микропроцессора. Классификация команд. Адресация памяти. Сегментация памяти. Прерывания.

Программное обеспечение микропроцессорной системы. Системное программное обеспечение. Операционная система. Взаимодействие прикладных программ с операционной системой.

2.1.2. Цели и задачи:

1) изучение состава микропроцессорной системы и ее основных частей;

2) ознакомление с типовой структурой микропроцессора и его системой команд;

3) ознакомление с назначением и составом системного программного обеспечения.

2.1.3. Ключевые вопросы:

1) микропроцессорная система;

2) типы микропроцессоров;

3) виды памяти микропроцессорной системы;

3) устройства ввода-вывода;

4) регистры микропроцессора;

5) основные виды команд микропроцессора;

6) операционная система;

2.1.4. Ссылки на литературные источники, приведенные в рабочей программе:

основная литература: 1,2;

дополнительная литература: ;

Интернет-ресурсы: 1,2,4.

2.1.5. Выводы по теме

Микропроцессорная система состоит из микропроцессора, памяти и устройств ввода-вывода. В зависимости от архитектуры и назначения выделяют несколько типов микропроцессоров. В процессе работы микропроцессор обменивается информацией с памятью, получает данные от устройств ввода и управляет устройствами вывода. Рассмотрены различные виды памяти, способы взаимодействия микропроцессора с устройствами ввода-вывода, а также система внутренних шин, связывающая указанные компоненты системы.

При выполнении программы микропроцессор активно использует свой внутренние регистры памяти. Часть регистров программно доступны и используются непосредственно прикладной программой (регистры общего назначения), другая часть предназначена для управления работой самого микропроцессора (системные регистры). Программа микропроцессора представляет собой набор инструкций, каждая из которых принадлежит определенной группе (команды пересылки, арифметические операции и т.д.). При работе с памятью

микропроцессор использует такой механизм, как сегментация памяти, т.е. выделение отдельных сегментов для команд, данных, другой информации. Для взаимодействия с устройствами ввода вывода и изменения порядка выполнения программы могут использоваться прерывания.

Программное обеспечение микропроцессорной системы разделяется на системное (операционная система) и прикладное. Операционная система организует взаимодействие между пользователем, прикладными программами и аппаратурой микропроцессорной системы.

2.2. Основы программирования на языке ассемблера (на примере MASM)

2.1.1. План лекции

Основы ассемблера. Директивы определения данных. Директивы определения байтов, слов и двойных слов. Определение массивов и их инициализация.

Команды. Общий формат команды: метка, мнемокод, операнды, комментарий. Примеры команд. Использование ссылок в командах: косвенная, индексная и базово-индексная адресации. Примеры. Задание размеров непосредственных операндов.

Команды пересылки, записи операндов в стек и извлечения из стека, команды ввода-вывода. Команды арифметических операций.

Переходы. Команды организации переходов. Безусловный переход. Условный переход: реализация типовой конструкции if-else на ассемблере.

Циклы. Команды организации циклов. Реализация типовых конструкций for и while на ассемблере.

Команды прерываний.

Директивы подготовки сегментов. Необходимость сегментации. Оформление сегментов. Директива ASSUME. Общая структура программы на языке ассемблера: сегменты стека, данных и кода. Точка входа в программу.

Подпрограммы на ассемблере. Команды вызова подпрограмм и возврата из подпрограмм. Оформление подпрограмм. Передача параметров подпрограмме и возвращение результата. Сохранение регистров в стеке.

Макросредства ассемблера: назначение, обзор возможностей.

2.2.2. Цели и задачи:

1) ознакомление с основами языка Assembler;

2) ознакомление со структурой прикладной программы при ее размещении в памяти;

3) ознакомление с принципами реализации типовых управляющих конструкций языков высокого уровня на языке Assembler с применением системы инструкций микропроцессора.

2.2.3. Ключевые вопросы:

1) составные части языка Assembler;

2) формат инструкции языка Assembler;

3) способы адресации данных в командах;

4) команды управления программой (переходов, организации циклов, вызовов подпрограмм);

4) макросредства Assembler.

2.2.4. Ссылки на литературные источники, приведенные в рабочей программе:

основная литература: 2,3;

дополнительная литература: - ;

Интернет-ресурсы: 1,2,3,4.

2.2.5. Выводы по теме

Язык Assembler по существу аналогичен языку инструкций микропроцессора, но позволяет использовать мнемонические коды команд вместо их цифровых кодов. Кроме того, в Assembler используются различные директивы, управляющие размещением данных в памяти и организации сегментов. Общий формат команды включает метку, мнемокод, операнды и комментарий. Оформление операндов специальным образом позволяет использовать различные виды адресации данных в памяти. Существуют команды, управляющие порядком вы-

полнения программы. Они выполняют безусловные и условные переходы, организуют циклы, вызовы подпрограмм и прерывания. Рассмотрена реализация типовых конструкций языков высокого уровня на языке ассемблера.

Общая структура программы на языке ассемблера подразумевает наличие сегментов стека, данных и кода. Эти сегменты подготавливаются с использованием специальных директив.

2.3. Алгоритмические языки программирования (на примере С++)

2.3.1. План лекции

Языки высокого уровня: общая характеристика. История развития языков высокого уровня. Четыре поколения языков высокого уровня.

Разработка программных продуктов с помощью языков высокого уровня. Этапы разработки. Препроцессорная обработка. Трансляция и интерпретация. Компиляция и компоновка.

Лексические основы языка высокого уровня (С++).

Идентификаторы и служебные слова. Константы: десятичные, восьмеричные, шестнадцатеричные, перечислимые, вещественные, символьные, строковые. Знаки операций. Унарные операции. Бинарные операции: аддитивные, мультипликативные, сдвига, поразрядные, отношения, логические, присваивания, выбора компонентов, обращения к элементам массива, условная, приведения типа, динамического распределения памяти. Ранги операций и их ассоциативность.

Разделители. Порядок применения (чтения) разделителей.

Данные как объекты памяти ЭВМ.

Понятие типа данных. Типизация. Основные типы данных языка С++. Производные типы данных: правила описания типов. Примеры производных типов. Определения и описание данных. Атрибуты типов памяти: класс памяти, видимость, продолжительность существования, тип компоновки. Примеры. Статические и динамические данные. Агрегатные типы данных. Массивы. Структурированные типы данных. Преобразования типов данных.

Операторы С++.

Простые и составные операторы. Операторы выбора. Конструкция if-else. Примеры применения. Переключатель switch-case. Примеры применения. Операторы цикла. Циклы с предусловием (while) и постусловием (do-while). Итерационный цикл (for). Операторы передачи управления.

Функции. Формат определения: тип возвращаемого значения, имя функции, список параметров. Прототипы функций.

2.3.2. Цели и задачи:

1) ознакомление с основными понятиями, классификацией и историей развития языков высокого уровня;

2) изучение лексических основ языка высокого уровня;

3) изучение основных типов данных языка высокого уровня;

4) изучение основных типов операторов языка высокого уровня;

2.3.3. Ключевые вопросы:

1) этапы разработки программы на языке высокого уровня;

2) лексические основы: идентификаторы, константы, операции, разделители;

3) типы данных и их атрибуты;

4) условный оператор, операторы, циклов, переключения, передачи управления;

4) функции и их прототипы.

2.3.4. Ссылки на литературные источники, приведенные в рабочей программе:

основная литература: 2;

дополнительная литература: 1, 6;

Интернет-ресурсы: 1,2,4.

2.3.5. Выводы по теме

Языки высокого уровня в своем развитии прошли четыре поколения от простейших языков для программирования математических вычислений до современных объектно-ориентированных языков. Разработка программы с помощью языков высокого уровня включает этапы препроцессорной обработки, трансляции и компоновки.

Лексические основы языка высокого уровня определяют состав используемых лексем (идентификаторы, константы, операции, разделители) и правила их использования.

Понятие типа данных является центральным для языка программирования. Он определяет объем памяти, отводимой под переменную, интерпретацию содержимого этой памяти и совокупность операций, разрешенных над данной переменной. Помимо типа данных каждая переменная имеет также определенный тип памяти, характеризующийся набором атрибутов: класс памяти, видимость, продолжительность существования, тип компоновки.

Языки высокого уровня помимо простых («одиночных») операторов включают специальные операторы управления, такие как операторы выбора, переключатели, операторы цикла и передачи управления. Кроме того, все языки поддерживают использование подпрограмм. В C++ подпрограммы называются функциями. Рассмотрены правила определения и вызовов функций.

2.4. Процедурное программирование

2.4.1. План лекции

Алгоритмическая декомпозиция программной системы.

Основные виды подпрограмм: процедуры и функции. Механизмы передачи параметров. Передача параметров по значению. Передача параметров по ссылке. Возврат результатов. Реализация передачи параметров по ссылке в C++: использование указателей и ссылок. Примеры. Возвращение ссылки. Пример. Определение, описание и вызов функций и процедур.

Разновидности подпрограмм в C++. Функции с начальными значениями параметров. Функции с переменным числом параметров: механизмы обращения к неявным параметрам. Рекурсивные функции: примеры. Подставляемые функции: достоинства и недостатки. Перегрузка функций. Шаблоны функций.

Структура программы процедурного типа. Принцип модульности. Разделение подпрограмм по модулям в сложных системах.

2.4.2. Цели и задачи:

1) ознакомление с содержанием понятия алгоритмической декомпозиции программной системы;

2) изучение механизмов передачи параметров подпрограмме и возврата результата в C++;

3) изучение основных разновидностей функций в C++;

4) освоение принципов построения программных систем при процедурном программировании.

2.4.3. Ключевые вопросы:

1) алгоритмическая декомпозиция программной системы;

2) передача параметров подпрограмме по значению и по ссылке;

3) реализация передачи параметров по ссылке в C++ с использованием указателей и ссылок;

4) функции с переменным числом параметров: адресация неявных параметров;

5) подставляемые функции;

6) рекурсивные функции;

7) перегрузка функций и механизм шаблонов;

8) принцип модульности.

2.4.4. Ссылки на литературные источники, приведенные в рабочей программе:

основная литература: 1,2;

дополнительная литература: 4,5;

Интернет-ресурсы: 1,2,3.

2.4.5. Выводы по теме

Алгоритмическая декомпозиция программной системы предполагает ее разделение на относительно независимые алгоритмы, которые реализуются подпрограммами. Подпрограммы получают исходные данные для алгоритмов и возвращают результаты их выполнения через механизмы передачи параметров. Помимо параметров подпрограммы в основном оперируют своими локальными переменными, однако могут обращаться и глобальным, общим для всех подпрограмм, переменным.

В C++ подпрограммы реализованы в виде функций, принимающих в общем случае несколько входных параметров и возвращающих один выходной. Входные параметры передаются функциям по значению, однако есть возможность организации передачи параметров по ссылке с использованием указателей и собственно «ссылок». Допускаются различные разновидности функций: с переменным числом параметров, рекурсивные, подставляемые. Под перегрузкой функций понимается использование в программе различных функций с одним и тем же именем, различающихся числом и типами входных параметров. Развитие идеи перегрузки состоит в использовании шаблонов, который вводит параметризированные типы и подразумевает выбор функции компилятором по типам фактических параметров при ее вызове.

Сложные программные системы строятся по модульному принципу. Модули процедурно-ориентированной программы содержат наборы связанных взаимными вызовами подпрограмм. Модули строятся таким образом, что внешние связи (вызовы подпрограмм из другого модуля) намного «слабее», чем внутренние (вызовы подпрограмм из того же модуля). Это позволяет ограничить интерфейсы модулей и раздельно их компилировать. В конечном итоге разделение программы на модули способствует организации совместной работы большой группы разработчиков и программистов.

2.5. Объектно-ориентированное программирование

2.5.1. План лекции

Проблемы проектирования сложных программных систем.

Факторы, обуславливающие сложность программных продуктов. Признаки сложных систем. Принципы преодоления сложности. Объектно-ориентированная декомпозиция.

Объектная модель и ее эволюция. Объектно-ориентированные анализ, проектирование и программирование. Составные части объектного подхода. Абстрагирование. Типы абстракций. Инкапсуляция: интерфейс и реализация класса. Модульность. Иерархия. Типизация. Параллелизм.

Реализация принципов абстрагирования и инкапсуляции в C++.

Классы и объекты. Определение классов. Компоненты классов. Статус доступа к компонентам классов. Определение компонентных функций. Конструкторы и деструкторы. Статические компоненты.

Состояние и поведение объектов классов.

Определение объектов. Идентичность объекта. Отношения между объектами. Жизненный цикл объектов.

Отношения между классами.

Основные типы отношений. Ассоциация и агрегация: примеры. Отношение использования: концепция клиент – сервер, подходы к обеспечению видимости сервера для клиента.

Наследование классов. Определение производных классов. Доступ компонентов базового класса в производном классе. Особенности конструкторов и деструкторов при наследовании. Множественное наследование и виртуальные базовые классы.

Полиморфизм. Механизм виртуальных функций. Позднее связывание. Абстрактные классы. Примеры.

Шаблоны классов.

2.5.2. Цели и задачи:

1) ознакомление с предпосылками возникновения объектно-ориентированного подхода к проектированию программных средств;

2) ознакомление с содержанием понятия объектно-ориентированной декомпозиции программной системы;

3) изучение объектной модели и ее составных частей;

4) освоение технологий объектно-ориентированного программирования в C++.

2.5.3. Ключевые вопросы:

1) объектно-ориентированная декомпозиция программной системы;

2) объектно-ориентированные анализ, проектирование и программирование.

3) классы и объекты классов в C++;

4) отношения между классами: ассоциация, агрегация, использование, наследование;

5) полиморфизм и позднее связывание;

6) параметризированные классы.

2.5.4. Ссылки на литературные источники, приведенные в рабочей программе:

основная литература: 1,2;

дополнительная литература: 1,2,6;

Интернет-ресурсы: 1,2,3,4.

2.5.5. Выводы по теме

Объектно-ориентированная декомпозиция сложной программной системы есть ее представление в виде совокупности взаимодействующих между собой объектов. Объекты обладают идентичностью, содержат данные и предлагают методы для обработки этих данных, а также реализации своего «внешнего» поведения. Объектную модель используют объектно-ориентированные анализ, проектирование и программирование. Рассматриваются составные части объектного подхода: абстрагирование, инкапсуляция, модульность, иерархия, типизация и параллелизм.

Реализация принципов абстрагирования и инкапсуляции в C++ предполагает разработку классов объектов. Компоненты классов имеют различные статусы доступа. Компонентами классов могут быть переменные и функции (методы). Среди последних выделяют специальные методы – конструктор и деструктор, вызываемые автоматически при создании и уничтожении объекта.

Отношения между объектами повторяют отношения между их классами. К основным их видам относят ассоциацию, агрегацию, использование и наследование. Ассоциацией называется смысловая связь различного рода между классами. Агрегация предполагает, что объект одного класса содержит внутри себя объект другого класса. Использование описывается в терминах технологии клиент-сервер. Под наследованием понимается отношение, когда класс-потомок наследует данные и методы класса-предка. Наследование позволяет создавать иерархию классов, что значительно упрощает повторное использование и модификацию исходных кодов программы.

Полиморфизм выводит технологию наследования на этап выполнения программы. При полиморфном вызове, реализуемом посредством механизма виртуальных функций, метод выбирается при выполнении уже скомпилированной программы в зависимости от типа реального объекта. В исходном тексте программы этот тип неизвестен, а сам объект адресуется указателем типа базового класса. Абстрактные классы описывают общее для всех их потомков строение и поведение, причем последнее представлено в виде «чистых» виртуальных функций. Эти функции сами по себе «ничего не делают» и представляют собой интерфейсы, которые обязаны реализовать классы-потомки. Объект любого из этих классов может быть адресован указателем типа абстрактного класса. Вызов виртуального метода через данный указатель будет распознаваться на этапе выполнения программы. Полиморфизм и абстрактные классы позволяют создавать гибкие программные конструкции.

Механизм шаблонов представляет параметризированные классы.

2.6. Программные данные и алгоритмы

2.6.1. План лекции

Сложные структуры данных – коллекции.

Классификация коллекций. Линейные и нелинейные коллекции.

Линейные коллекции с прямым доступом: массивы, записи, файлы.

Линейные коллекции с последовательным доступом: связные списки, стеки, очереди, очереди приоритетов.

Линейные коллекции с индексным доступом: словари, hash-таблицы.

Нелинейные иерархические коллекции: деревья и их разновидности, пирамиды (heap-деревья).

Групповые нелинейные коллекции: множества, графы, сети.

Ввод-вывод данных.

Потоки ввода-вывода. Общая структура обмена данными через потоки. Входные, выходные и двунаправленные потоки. Структура классов библиотеки потокового ввода-вывода (основные классы). Стандартные потоки. Объекты классов стандартных потоков. Перегрузка операций ввода-вывода в стандартные потоки для пользовательских типов. Форматирование данных при обмене с потоками: флаги класса ios и функции-манипуляторы. Функции для работы с потоками.

Файлы. Типы файлов. Работа с файлами: последовательность действий, примеры.

Основные принципы и подходы к проектированию программных алгоритмов.

Классы алгоритмов. Критерии оценки вычислительных алгоритмов. Системная эффективность. Эффективность пространства. Вычислительная эффективность. Определение вычислительной эффективности алгоритма сортировки сравнением по числу сравнений и числу обменов. Лучший, худший и средний случаи. Порядок алгоритма: определение. Порядок алгоритма сортировки сравнением по числу сравнений. Сортировка выбором: определение порядка алгоритма.

Методы проектирования алгоритмов. Методы частных целей, подъема ветвей и границ, эвристический метод. Примеры

Виды алгоритмов. Итерационные и рекурсивные алгоритмы, особенности их программной реализации, преимущества и недостатки. Примеры.

Методы сортировки и поиска данных. Методы линейного и бинарного поиска. Реализация на C++. Вычислительная эффективность.

Методы сортировки. Сортировка вставками. Реализация на C++. Анализ вычислительной эффективности. Пузырьковая сортировка. Реализация на C++. Анализ вычислительной эффективности. Быстрая сортировка. Реализация на C++. Анализ вычислительной эффективности. Сравнение алгоритмов сортировки.

2.6.2. Цели и задачи:

1) изучение классификации коллекций;

2) изучение классы и критериев оценки вычислительных алгоритмов;

3) освоение технологий файлового ввода-вывода;

4) изучение методов проектирования алгоритмов;

5) изучение алгоритмов поиска и сортировки.

2.6.3. Ключевые вопросы:

1) линейные коллекции с прямым, последовательным и индексным доступом;

2) иерархические (деревья, пирамиды) и групповые (множества, графы, сети) нелинейные коллекции;

3) стандартные потоки ввода-вывода;

4) чтение и запись файлов;

5) вычислительная эффективность и порядок алгоритма;

6) методы частных целей, подъема ветвей и границ, эвристический метод;

7) итерация и рекурсия;

8) линейный и бинарный поиск;

9) методы сортировки: сравнением, выбором, вставками, пузырьковая;

10) быстрая сортировка.

2.6.4. Ссылки на литературные источники, приведенные в рабочей программе:

основная литература: 2;

дополнительная литература: 2,3,5;

Интернет-ресурсы: 1,2,3.

2.6.5. Выводы по теме

Коллекции представляют собой собрания однотипных данных. По структуре они разделяются на линейные и нелинейные. Линейные коллекции могут иметь прямой, последовательный и индексный доступ. В последнем случае индекс элемента в массиве определяется hash-функцией по ключу. Нелинейные коллекции делятся на иерархические (деревья) и групповые (множества, графы, сети).

Файлы могут рассматриваться как коллекции с прямым или последовательным доступом. Файловый ввод-вывод данных прикладной программы может быть организован с привлечением потоков ввода-вывода. Потоки могут быть входными, выходными и двунаправленными. Они реализуются классами стандартной библиотеки C++, скрывающими от программиста низкоуровневые функции операционной системы. Работа с файлами может быть организована как с использованием стандартных классов потоков, так и с применением стандартных библиотечных функций.

Для работы с коллекциями применяются алгоритмы удаления и вставки, поиска и сортировки. Существуют различные критерии оценки алгоритмов. Вычислительная эффективность алгоритма определяется функцией числа элементарных операций (сравнения, копирования, обмена) от числа элементов коллекции. В зависимости от вида этой функции определяется порядок алгоритма.

Существует несколько основных методов проектирования алгоритмов. Рассматриваются методы частных целей, подъема ветвей и границ, эвристический метод. Приводится сравнение итерационные и рекурсивных алгоритмы, анализируются особенности их программной реализации, преимущества и недостатки.

Известны два метода поиска (последовательный и бинарный) и множество различных методов сортировки. Рассматриваются и сравниваются методы сортировки порядка n^2 (сравнением, выбором, вставками, пузырьковая) и порядка nlog(n) (быстрая). Приводится их программная реализация и анализ вычислительной эффективности по числу сравнений и обменов.

2.7. Проектирование программных систем

2.7.1. План лекции

Этапы проектирования сложных программных систем. Микро- и макропроцессы проектирования, их отношения.

Этапы макропроцесса:

концептуализация – установление основных требований к системе;

анализ – создание модели поведения системы;

проектирование – создание архитектуры реализации;

эволюция – последовательное приближение системы к желаемому результату);

сопровождение – управление эволюцией системы в ходе ее эксплуатации.

Этапы микропроцесса:

идентификация классов и объектов на данном уровне абстракции;

выявление семантики классов и объектов;

выявление связей между классами и объектами;

реализация классов и объектов.

Стандарты на разработку прикладных программных средств.

Практические вопросы: распределение ресурсов, роли разработчиков, интеграция процесса, управление конфигурацией и версиями, тестирование. Повторное использование и качество программного продукта. Документация проекта. Инструменты проектирования. Критерии оценки проекта.

Эксплуатация программных средств.

2.7.2. Цели и задачи:

1) изучение этапов проектирования сложных программных средств;

2) ознакомление со стандартами на разработку прикладных программных средств;
3) ознакомление с практическими вопросами проектирования.
2.7.3. Ключевые вопросы:
1) этапы макропроцесса проектирования;
2) этапы микропроцесса проектирования;
3) стандарты на разработку прикладных программных средств;
4) управление ресурсами разработки;
5) сопровождение и эксплуатация программных средств.
2.7.4. Ссылки на литературные источники, приведенные в рабочей программе:
основная литература: 2,3;
дополнительная литература: 3, 5;
Интернет-ресурсы: 1,2,4.
2.7.5. Выводы по теме

Проектирование сложных программных систем выполняется как на уровне макропроцесса, та и на уровне микропроцесса. На уровне макропроцесса устанавливаются основные требования к системе, создаются модели ее поведения, архитектура реализации, проводится последовательное приближение системы к желаемому результату, происходит управление эволюцией системы в ходе ее эксплуатации. На уровне микропроцесса производится идентификация классов и объектов на данном уровне абстракции, выявляются семантики классов и объектов, связи между ними, происходит реализация классов и объектов. Принят ряд стандартов на разработку прикладных программных средств.

Рассматриваются практические вопросы проектирования, связанные с распределением ресурсов и роле разработчиков, интеграцией процесса, управлением конфигурацией и версиями, тестированием. Программный продукт должен допускать повторное использование и обладать необходимым качеством. Крайне важна качественная документация проекта. Современные инструменты проектирования (интегрированные среды разработки) могут в значительной степени помочь в решении практических вопросов проектирования.

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ (РЕКОМЕНДАЦИИ)

3.1. Методические указания по изучению дисциплины

1) Следует тщательно планировать и организовывать время, необходимое для изучения дисциплины. Недопустимо откладывать ознакомление с теоретической частью, подготовку отчетов к лабораторным работам и выполнение курсовой работы на конец семестра, поскольку это неминуемо приведет к снижению качества освоения материала, оформления отчетов и работы. Все виды работ по дисциплине рекомендуется выполнять по календарному плану, приведенному в Рабочей программе.

2) «Сценарий изучения дисциплины» предусматривает следующие схемы:

по теоретическому курсу: ознакомление с тематикой лекции в разделе «Краткое изложение программного материала» → изучение литературы по теме → прослушивание лекции и обсуждение вопросов;

по выполнению практических и лабораторных работ:

подготовка к выполнению работы по учебному пособию (изучение теоретических сведений, разработка макетов программ, планирование работ) → работа на практическом занятии (коллективное выполнение заданий по отдельным вариантам под контролем преподавателя) → разработка программы по собственному варианту (подготовка исходного кода программы, компиляция, компоновка, тестирование) → доработка программы на лабораторном занятии в компьютерном классе и демонстрация компиляции, компоновки и тестирования преподавателю → подготовка отчета по работе → защита лабораторной работы;

по выполнению курсовой работы: получение и уточнение задания → разработка компьютерной программы → демонстрация работы программы преподавателю → устранение выявленных недостатков → демонстрация окончательного варианта → оформление курсовой работы → защита курсовой работы.

3) Материалы учебно-методического комплекса для студентов являются обязательными к ознакомлению, поскольку являются «отправной точкой» для изучения дисциплины. В разделе «Краткое изложение программного материала» приведены тематика лекционных занятий, планы, цели и задачи лекций, ключевые вопросы и выводы, а также ссылки на литературу. Ознакомившись с разделом, студент получает возможность самостоятельно подготовиться к лекции, изучив теоретический материал, а непосредственно на занятии – занимать активную позицию, задавая вопросы лектору и вступая в дискуссии по теме. В разделе «Методические указания (рекомендации)» приведены указания к выполнению практических и лабораторных работ и курсовой работы, а также самостоятельной работы. Изучив материал раздела, студенты получают возможность грамотно планировать выполнение всех видов работ, выполнять работы в соответствие со всеми приведенными требованиями, подготавливать отчеты и оформлять работы. В разделе «Контроль знаний» приведены материалы, которые позволят студентам подготовиться к процедурам текущего контроля (тестирование в рамках проведения «контрольных точек») и итогового контроля (экзамен).

4) Изучение дисциплины требует непрерывной работы с литературой. Перед прослушиванием каждой лекции студент должен ознакомиться с материалом по списку, приведенному по теме лекции в разделе «Краткое изложение программного материала». Перед выполнением практических и лабораторных работ необходимо изучить теоретические сведения, приведенные в учебном пособии и выполнить все требуемые в плане подготовке к работе операции.

5) При подготовке к экзамену следует придерживаться следующих рекомендаций:

подготовку к экзамену нужно проводить в течение всего курса изучения дисциплины. После предварительного изучения теоретического материала перед прослушиванием лекции следует составить планы ответа на каждый экзаменационный вопрос по теме лекции. После прослушивания лекции эти планы при необходимости уточняются с учетом изменения представлений. Окончательная корректировка планов ответов производится уже после изучения всего курса, когда устанавливаются и осознаются связи между всеми разделами и темами;

при подготовке к экзамену следует полностью исключить все виды «заучивания» материала, основанные на «механической» фиксации фонетической или аудиовизуальной информации в памяти. Вместо этого основной упор следует сделать на раскрытие причинно-следственных связей, логических закономерностей и общих тенденций;

необходимо правильно организовать процесс подготовки к экзамену на сессии как в плане чередования труда и отдыха, так и в плане организации занятий. На первом этапе подготовки (за 2-3 дня до экзамена) следует выполнить «общий обзор» курса с целью выделения «простых» и «сложных» тем. Далее нужно сделать упор на освоение и уточнение наиболее сложных вопросов. И, наконец, непосредственно накануне экзамена нужно еще раз сделать «общий обзор» с целью систематизации полученных знаний. Таким образом, график изменения интенсивности занятий должен иметь участки увеличения, стабилизации на максимуме и снижения. Это позволяет подойти к экзамену в наилучшей физической и психологической форме.

6) При работе с тестовой системой курса необходимо руководствоваться следующим. Тесты ни в коем случае не следует рассматривать «самодостаточными» в том смысле, что абсолютно неверно представление о том, что правильно выполненный тест является свидетельством полного освоения материала. Тестовые вопросы должны рассматриваться в первую очередь как указатели направления интеллектуальных усилий по установлению связей между теоретическими положениями, практическими вопросами, примерами и т.д. Поэтому не следует «механически» запоминать правильные ответы на тестовые вопросы, тем более что практика проведения контрольных мероприятий по дисциплине предусматривает допол-

нения тестовой процедуры уточняющими вопросами преподавателя, призванными выявить аргументацию ответов студента. Вместо заучивания следует добиваться понимания сути вопроса, построения логических цепочек, обосновывающих ответ с привлечением теоретических положений.

Перечень учебно-методических изданий, рекомендуемых студентам для подготовки к занятиям и выполнению самостоятельной работы, приведен в Рабочей программе.

3.2. Методические указания к практическим занятиям

1) План проведения занятий с указанием последовательности изучаемых модулей, тем занятий, объема аудиторных часов, отводимых для освоения материалов по каждой теме, а также часов для самостоятельной работы студентов приведен в Рабочей программе.

2) Теоретические положения и указания к выполнению практических работ приведены в учебном пособии:

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.

3) На практическом занятии производится разбор и коллективное выполнение выбранных вариантов заданий методом мозгового штурма. При этом преподавателем выбирается либо незадействованные варианты, либо варианты повышенной сложности. У доски работает студент по его желанию, вызванный преподавателем или по варианту разбираемой задачи.

Для целенаправленной работы студента в ходе подготовки к практическому занятию необходимо использовать следующие издания:

Валюк, Т.В. Программирование на СИ/СИ++: учеб. - метод. пособие / Т. В. Валюк, Л. А. Соловцова ; АмГУ, ФМиИ. - Благовещенск : Изд-во Амур. гос. ун-та, 1999. - 62 с.

3.3. Методические указания к лабораторным занятиям

1) План проведения занятий с указанием последовательности изучаемых модулей, тем занятий, объема аудиторных часов, отводимых для освоения материалов по каждой теме, а также часов для самостоятельной работы студентов приведен в Рабочей программе.

2) Теоретические положения и указания к выполнению лабораторных работ приведены в учебных пособиях:

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.

3) Методические вопросы, связанные с подготовкой и проведением лабораторных занятий

Подготовка к лабораторной работе включает:

изучение теоретических сведений, приведенных в учебном пособии;

разработка программных алгоритмов и макетов программ;

работа на практическом занятии по теме (возможно, частичное выполнение задания по варианту);

самостоятельную работу по выполнению задания после практического занятия (подготовка исходного кода программы, компиляция, компоновка, тестирование).

Непосредственно на лабораторном занятии студенты

окончательно дорабатывают программный код, компилируют, компонуют и тестируют программу под контролем преподавателя;

демонстрируют работу программ и процедуры тестирования преподавателю.

После выполнения работы окончательно формируется отчет. Отчет должен содержать:

задание;

исходный код программы без комментариев;

результаты тестирования.

По ряду работ и вариантов отчет по желанию студента может быть дополнен

спецификацией программы;

блок-схемами алгоритмов.

Защита лабораторных работ проводится в форме собеседования. Цель собеседования выявить понимание студентом теоретического материала, примененных алгоритмов и программных конструкций.

4) Рекомендации по организации рабочего места студента, соблюдению правил техники безопасности и санитарных норм

Лабораторные работы проводятся в компьютерном классе. Требования к организации рабочего места студента, соблюдению правил техники безопасности и санитарных норм являются типовыми для аудиторий, предназначенных для работы с персональными компьютерами.

Каждый студент должен работать за отдельным компьютером. Не допускается совместное выполнение работ двумя или более студентами за одним компьютером.

На первом лабораторном занятии проводится инструктаж студентов по технике безопасности. Проведение инструктажа обязательно фиксируется в специальном журнале подписями инструктирующего (преподавателя) и инструктируемых (студентов).

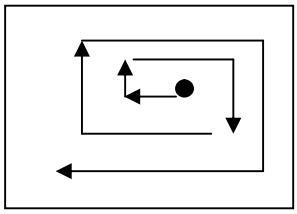
3.4. Методические указания по выполнению курсовой работы

1) Тематика курсовых работ

Выполнение курсовой работы состоит в разработке несложной компьютерной программы в соответствии с вариантами

Варианты заданий

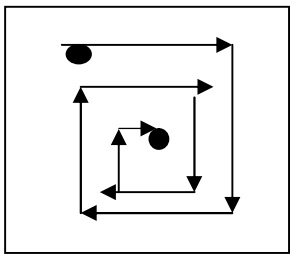
№1 Игра «Прыжок кенгуру»



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»

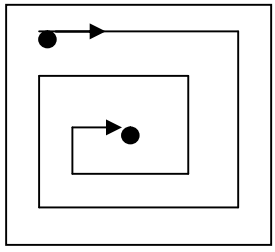


В левом верхнем углу поля 5*5 позиций (можно взять поле другого нечетного размера) стартует заяц. Он прыгает по спирали до центра. Прыгать заяц может либо в соседнюю клетку, либо через одну (случайным образом)

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:

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

Б) 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;

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;

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 для 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:

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

Б) 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:

А) число особей поколения;

Б) число поколений;

В) 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.

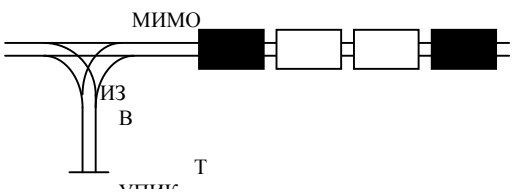
Указание:

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.

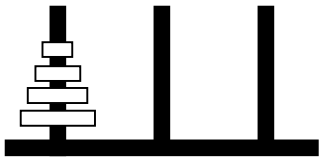
№34 Программа «Железнодорожный узел»



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 сто-

роне 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

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

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

В) 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.

Г) Расшифровать текст №1, используя следующую процедуру:

составить таблицу встреч символов и отсортировать ее;

сопоставив таблицы, заменить символы в тексте.

2) Методические указания по выполнению курсовой работы

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.

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;

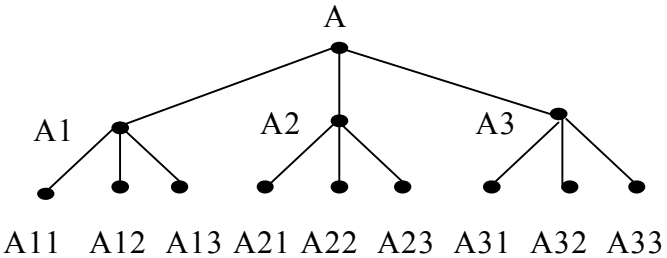
пользовательский интерфейс должен предоставлять пользователю справочную информацию.

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).

Согласование результатов внешнего проектирования с руководителем курсовой работы

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

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 зацикliва-

ний и недостижимых частей) можно написать с использованием только следующих логических структур:

последовательности двух или более операторов;

выбора одного из двух операторов;

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.

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 про-*

граммы. 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 тесты, 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 каж-

дого 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.

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.

По согласованию с преподавателем некоторые разделы могут быть опущены.

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) Защита курсовой работы

Защита курсовой работы проводится в форме собеседования. Цель собеседования выявить понимание студентом теоретического материала, примененных алгоритмов и программных конструкций.

Критерии оценки курсовой работы:

Оценка «удовлетворительно» ставится, если

основные результаты работы, не являясь наилучшими из возможных, все же удовлетворяют предъявляемым требованиям;

в результате собеседования выявлено понимание студентом основных положений теории, использованной при выполнении работы, однако ряд частных положений остался не проясненным.

Оценка «хорошо» ставится, если

основные результаты работы близки к оптимальным, однако ответы на вопросы преподавателя выявили неполное понимание теоретических положений;

ответы на вопросы преподавателя выявили полное понимание теоретических положений, однако результаты работы, удовлетворяя в целом предъявляемым требованиям, далеки от оптимальных.

Оценка «отлично» ставится, если

студентом получены результаты, близкие к оптимальным;

в результате собеседования выявлено понимание студентом всех положений теории, использованной при подготовке работы.

3.5. Методические указания по самостоятельной работе студентов

1) Самостоятельная работа студентов по дисциплине предусматривает выполнение курсовой работы, а также подготовку к выполнению практических и лабораторных работ и оформление отчетов по ним. Общая схема СРС приведена в Рабочей программе.

2) Рекомендации по планированию и организации времени, отведенного на изучение дисциплины, приведены в разделе 3.1. «Методические указания по изучению дисциплины».

3) Перечень тем теоретического курса, предварительное изучение которых необходимо для выполнения практических и лабораторных работ:

Все теоретические сведения, необходимые для выполнения работ, содержатся в учебных пособиях к их выполнению.

4) Рекомендации по работе с литературой приведены в разделе 3.1. «Методические указания по изучению дисциплины».

5) Рекомендации по подготовке к экзамену приведены в разделе 3.1. «Методические указания по изучению дисциплины».

6) рекомендации по отдельным видам работ при освоении дисциплины приведены в разделе 3.1. «Методические указания по изучению дисциплины».

7) Рекомендации по подготовке отчетов о выполнении СРС.

«Отчетами» о выполнении СРС являются

курсовая работа по дисциплине;

отчеты о выполнении лабораторных работ.

Выполнение курсовой работы регламентируется разделом 3.4. «Методические указания по выполнению курсовой работы».

Требования к содержанию отчетов о выполнении лабораторных работ приведены в разделе 3.3. «Методические указания по выполнению лабораторных работ»

4 КОНТРОЛЬ ЗНАНИЙ

4.1 Текущий контроль знаний

Текущий контроль знаний предусматривает две контрольных точки. Оценка по контрольным точкам выставляется на основании результатов

выполнения практических и лабораторных работ, прошедших к моменту контрольной точки;

тестирования по разделам теста, приведенного в п. 4.3. Собственно процедура тестирования дополняется собеседованием преподавателя со студентом по вопросам теста с целью выяснения аргументации последнего.

Итоговая оценка студента формируется преподавателем, ответственным за дисциплину (лектором) по данным, предоставленным преподавателем, ведущим практические и лабораторные занятия и преподавателем, проводившим тестирование.

4.2. Итоговый контроль знаний

Итоговый контроль знаний подразумевает экзамен по теоретическому курсу.

Экзамен предусматривает письменные ответы студента на два теоретических вопроса из списка вопросов, приведенного в Рабочей программе.

Для получения удовлетворительной оценки на экзамене достаточно показать знание основных понятий по теме экзаменационных вопросов.

Оценка «хорошо» выставляется студенту, показавшему способность алгоритмического, математического, технического и др. обоснований решений в своих письменных ответах на экзаменационные вопросы.

Оценка «отлично» выставляется студенту, выполнившему все требования, приведенные выше, и правильно ответившему на дополнительные вопросы по смежным темам. При этом неправильные ответы на дополнительные вопросы могут служить основанием для снижения оценки до «удовлетворительно», если эти ответы свидетельствуют о слабом понимании материала.

4.3. Тестовые вопросы

Приведенные вопросы используются для компьютерного тестирования знаний студентов специальности 220301 «Автоматизация технологических процессов и производств» по дисциплине «Программирование и основы алгоритмизации».

Вопросы охватывают основные темы, изучаемые студентами в данном курсе, и сгруппированы по разделам:

1. Состав вычислительной системы. Построение программ.

2. Базовые понятия языка программирования C++.

3. Объектно-ориентированное программирование.

4. Структуры данных и алгоритмы.

Тестирование является составной частью процедуры промежуточного контроля знаний (в ходе изучения дисциплины), а также используется для контроля остаточных знаний (после окончания изучения дисциплины).

Тестирование проводится с привлечением программы Test Maker. Тест, применяемый при итоговом контроле знаний, включает 25 вопросов, в число которых попадает фиксированное количество вопросов из каждого раздела. Вопросы из списка выбираются случайным образом, ответы также приводятся в случайном порядке. Часть вопросов имеет один вариант правильного ответа, остальные – несколько. О числе правильных вариантов (один/несколько) сообщается. Выставление оценок осуществляется программой по настраиваемым правилам. Рекомендуются следующие правила: ответ на вопрос принимается как верный, если выбраны все правильные ответы, оценка «удовлетворительно» – 70% правильных ответов, оценка «хорошо» – 80% правильных ответов, «отлично» – 90% правильных ответов.

Раздел 1. Состав вычислительной системы. Построение программ.

1. Системный стек не используется для
   1) хранения адреса возврата из подпрограммы;
   2) размещения параметров подпрограмм;
   3) размещения локальных переменных подпрограмм;
   4) хранения кода подпрограмм.

2. Способом трансляции программ не является
   1) кодирование;
   2) интрепретирование;
   3) ассемблирование;
   4) компилирование.

3. Преимуществом интерпретирования не является
   1) возможность оперативной коррекции программы;
   2) межплатформенная совместимость;
   3) скорость выполнения программы;
   4) безопасность работы программы.

4. Какой язык программирования позволяет создавать наиболее эффективные (с точки зрения скорости, объема кода и данных) программы?
   1) Pascal;
   2) C++;
   3) Basic;
   4) Assembler.

5. Библиотечные функции включаются в программу на стадии

1) Редактирования;
2) препроцессорной обработки;
3) компиляции;
4) компоновки.

6. В результате компиляции создаются
1) исходные файлы;
2) объектные файлы;
3) исполняемый файлы;
4) файлы заголовков.

7. Связывание модулей программы осуществляется на этапе
1) редактирования;
2) препроцессорной обработки;
3) компиляции;
4) компоновки.

8. Вещественные числа в современных ПК
1) представляются в формате с фиксированной точкой;
2) представляются в формате с плавающей точкой;
3) эмулируются программно с помощью целых;
4) не используются.

Раздел 2. Базовые понятия языка программирования С++

1. Функции в С/С++ имеют
1) ограниченное число входных параметров и один выходной;
2) один входной и один выходной параметры;
3) неограниченное число входных параметров и один выходной;
4) один входной параметр и ограниченное число выходных.

2. Структура это объект, состоящий из
1) неименованных элементов различных типов;
2) неименованных элементов одного типа;
3) именованных элементов различных типов;
4) именованных элементов одного типа.

3. Указатель - это переменная,
1) содержащая число с плавающей точкой;
2) принимающая значения "истина" или "ложь";
3) содержащая адрес другой переменной или функции;
4) содержащая адрес разработчика программы.

4. Оператором С++ new создаются объекты
1) локальной продолжительности существования;
2) статической продолжительности существования;
3) динамической продолжительности существования;
4) другие.

5. Прототипы функций не применяются для
1) описания функций, определенных в других модулях;

2) описания функций, определенных ниже в данном модуле;
3) описания функций, определенных выше в данном модуле;
4) описания функций в файлах заголовков.

6. Обязательным для самостоятельной программы на C/C++ является
1) объявление статических переменных;
2) использование динамической памяти;
3) определение пользовательских классов;
4) наличие функции main (WinMain).

7. Переменные, определенные в другом модуле описываются с использованием ключевого слова
1) extern;
2) static;
3) auto;
4) void.

8. Об уничтожении каких переменных не нужно заботиться при написании программы на C/C++:
1) Динамических;
2) Автоматических;
3) Любых неинициализированных;
4) Любых инициализированных.

9. Соответствие идентификатора объекту в программе, размещенной в нескольких модулях, определяется таким атрибутом объекта как
1) класс памяти;
2) продолжительность существования;
3) видимость;
4) тип компоновки.

10. Имя массива в C/C++ это имя
1) константной ссылки на его первый элемент;
2) переменной-указателя, содержащей адрес первого элемента массива;
3) переменной, содержащей число элементов массива;
4) переменной-указателя, содержащей адрес последнего элемента массива.

11. Часть программы, по которой по идентификатору объекта можно получить доступ именно к этому объекту определяется таким атрибутом объекта как
1) класс памяти;
2) продолжительность существования;
3) видимость;
4) тип компоновки.

12. Автоматические переменные C++ уничтожаются
1) с помощью оператора delete;
2) при выходе из блока;
3) при повторном объявлении;
4) автоматически за ненадобностью.

13. Какой порядок использования объектов как параметров функции не предусматривает возможности их изменения после ее выполнения в C++?

1) Передача значения объекта;
2) Передача ссылки на объект;
3) Передача указателя на объект;
4) Передача ссылки на указатель на объект.

14. Тип функции C++ определяется
1) типом самого левого параметра;
2) типом самого правого параметра;
3) именем функции;
4) типом возвращаемого результата.

15. В программе C++ могут содержаться две функции с одинаковыми именами, если у них различны
1) тип возвращаемого результата;
2) спецификация формальных параметров;
3) значения по умолчанию;
4) операторы тел.

16. Подставляемые функции не отличаются от "обыкновенных"
1) в исполняемом коде, генерируемом компилятором;
2) в исходном коде оформлением определения;
3) в исходном коде при вызове;
4) нет вариантов.

17. Так же как и указатель, ссылка на объект в C++
1) имеет значение адреса объекта;
2) есть константа и не может быть настроена на другой объект;
3) автоматически "разыменовывается" при обращении к объекту;
4) требует обязательной инициализации при определении.

18. Прототип функции имеет вид: void fun(int &); Если x -имя переменной типа int, какой из вызовов функции будет правильным?
1) fun(&x);
2) fun(x);
3) fun(*x);
4) fun().

19. Тип объекта памяти языка C++ не определяет
1) объем памяти, выделяемый объекту;
2) продолжительность существования объекта;
3) совокупность разрешенных операций над объектом;
4) интерпретация содержимого памяти, выделенной объекту.

20. Формирование функции по шаблону выполняется
1) самой программой на этапе выполнения при распознавании вызова;
2) компилятором на этапе компиляции при распознавании вызова;
3) программистом при редактировании исходного текста;
4) пользователя программы по запросу от нее.

21. В левой части операции присваивания в C++ не могут стоять
1) имена массивов;

2) имена указателей;
3) имена скалярных переменных;
4) вызовы функций.

22. Переменная какого типа не может быть объявлена в программе на C/C++?
1) массив указателей;
2) массив указателей на массивы;
3) массив ссылок;
4) ссылка на массив.

23. Какой тип данных C++ не принадлежит к целым числам?
1) char;
2) int;
3) double;
4) long

24. Строка в C/C++ - это
1) массив данных типа char с нуль-символом в конце;
2) массив указателей на char с нуль-указателем в конце;
3) массив данных типа char с числом элементов в первом элементе;
4) нет правильного варианта.

25. Какой тип данных C++ не принадлежит к вещественным числам?
1) float;
2) long;
3) double;
4) long double.

26. Включение в программу текстов из файлов осуществляется с помощью директивы препроцессора
1) #define;
2) #include;
3) #ifdef;
4) #pragma.

27. В программе на C++ используется перечислимый тип enum T{one, two, three}; Какое значение имеет константа two?
1) 0;
2) 1;
3) 2;
4) неопределенное.

28. Замены в тексте программы осуществляется с помощью директивы препроцессора
1) #define;
2) #include;
3) #ifdef;
4) #pragma.

29. В конструкции int a = 10; элементы располагаются в последовательности
1) имя типа - имя переменной - знак операции – константа;
2) имя переменной - имя типа - знак операции - константа;
3) имя типа - имя переменной - разделитель – константа;

4) конструкция синтаксически неверна.

30. Условная компиляция фрагментов программы осуществляется с помощью директивы препроцессора
1) #define;
2) #include;
3) #ifdef;
4) все варианты верны.

31. Переменная p в программе имеет следующий тип: int (*p)[3]; На какое число увеличится значение p в результате операции p++ (sizeof(int) =2)?
1) 1;
2) 2;
3) 6;
4) 12.

32. Макроподстановки в тексте программы осуществляется с помощью директивы препроцессора
1) #define;
2) #include;
3) #ifdef;
4) нет правильных вариантов.

33. В программе имеется следующее определение: char M[] = {1, 2, 3, 4}; Каково значение выражения *(M++)?
1) 1;
2) 2;
3) 3;
4) синтаксическая ошибка.

34. В программе имеется следующее определение: char M[] = {1, 2, 3, 4}; Каково значение выражения M[2]?
1) 1;
2) 2;
3) 3;
4) синтаксическая ошибка.

35. В программе имеется следующее определение: char *s = "ABC".Какой объем памяти выделяется в соответствии с этим определением (размер памяти для указателя составляет 2 байта, для данного типа char - 1 байт)?
1) 2 байта;
2) 3 байта;
3) 5 байт;
4) 6 байт.

36. Какой из операторов C++ среди перечисленных является оператором цикла?
1) if;
2) else;
3) while;
4) switch.

37. Какой из операторов передачи управления в C++ означает переход к следующей

итерации?

1) goto;
2) return;
3) break;
4) continue.

38. Каждый отдельный оператор в C/C++ заканчивается знаком

1) "!";
2) "." ;
3) "@";
4) ";" .

39. Пользовательские классы объявляются в C/C++ с использованием ключевого слова

1) class;
2) struct;
3) union;
4) всех трех, приведенных выше.

40. Какой из операторов передачи управления C++ означает безусловный переход к метке?

1) goto;
2) return;
3) break;
4) continue.

41. Какой из операторов передачи управления C++ означает выход из цикла?

1) goto;
2) return;
3) break;
4) continue.

Раздел 3. Объектно-ориентированное программирование

1. Функция перегрузки бинарной операции сложения объявлена как дружественная функция в классе AnyClass. X и Y - объекты класса AnyClass. Каким образом будет вызвана функция для выполнения операции X + Y:

1) operator+(X,Y);
2) X.operator+(Y);
3) Y.operator+(X);
4) нет варианта.

2. Функция перегрузки унарной операции изменения знака определена как компонентная функция класса AnyClass. X - объект класса AnyClass. Каким образом будет вызвана функция для выполнения операции -X?

1) operator-(X);
2) operator-();
3) X.operator-();
4) X.operator-(X).

3. Функция перегрузки операции постинкремента определена как компонентная функция класса AnyClass. X - объект класса AnyClass. Каким образом будет вызвана функция для выполнения операции X++?

1) operator++(X);
2) X.operator++();
3) X.operator++(0);
4) operator++(X,0).

4. Функция перегрузки операции постинкремента объявлена как дружественная функция в классе AnyClass. X - объект класса AnyClass. Каким образом будет вызвана функция для выполнения операции X++:

1) operator++(X);
2) X.operator++();
3) X.operator++(0);
4) operator++(X,0);

5. Если один объект является частью другого, это пример отношения типа

1) ассоциация;
2) агрегация;
3) наследование;
4) использование.

6. Отношение вида "общее-частное" есть отношение типа

1) ассоциация;
2) агрегация;
3) наследование;
4) использование.

7. Компонент X базового класса имеет статус доступа private в этом классе. Производный класс наследует компоненты базового со спецификатором доступа public. Какой статус доступа будет иметь X в производном классе:

1) public;
2) protected;
3) private;
4) компонент будет недоступен функциям производного класса.

8. Компонент X базового класса имеет статус доступа public в этом классе. Производный класс наследует компоненты базового со спецификатором доступа private. Какой статус доступа будет иметь X в производном классе:

1) public;
2) protected;
3) private;
4) компонент будет недоступен функциям производного класса.

9. Компонент X базового класса имеет статус доступа public в этом классе. Производный класс наследует компоненты базового со спецификатором доступа public. Какой статус доступа будет иметь X в производном классе?

1) public;
2) protected;
3) private;
4) компонент будет недоступен функциям производного класса.

10. Компонент X базового класса имеет статус доступа protected в этом классе. Производный класс наследует компоненты базового со спецификатором доступа public. Какой

статус доступа будет иметь X в производном классе?

1) public;
2) protected;
3) private;
4) компонент будет недоступен функциям производного класса

11. Компонент X базового класса имеет статус доступа protected в этом классе. Производный класс наследует компоненты базового со спецификатором доступа private. Какой статус доступа будет иметь X в производном классе?

1) public;
2) protected;
3) private;
4) компонент будет недоступен.

12. Конструкторы и деструкторы базового и производного классов (б.к. и п.к.) вызываются в следующем порядке:

1) констр. б.к. - констр. п.к. - дестр. б.к. - дестр. п.к.;
2) констр. п.к. - констр. б.к. - дестр. б.к. - дестр. п.к.;
3) констр. б.к. - констр. п.к. - дестр. п.к. - дестр. б.к.;
4) констр. п.к. - констр. б.к. - дестр. п.к. - дестр. б.к.

13. Базовый класс объявляется виртуальным для того, чтобы

1) упростить описание объектов;
2) исключить повторное наследование при множественном наследовании;
3) сделать все функции класса виртуальными;
4) объявить все компоненты класса защищенными.

14. Выбор виртуальной функции при полиморфном вызове производится

1) на этапе редактирования исходного текста самим программистом;
2) на этапе компиляции компилятором;
3) на этапе компоновки компоновщиком;
4) на этапе выполнения самой программой.

15. Механизм виртуальных функций подразумевает, что

1) по указателю типа базового класса, настроенному на объект производного класса вызывается функция производного класса;
2) по имени объекта производного класса вызывается функция производного класса;
3) компонентная функция производного класса вызывается безотносительно объекта путем указания квалифицированного имени;
4) компонентная функция базового класса вызывается для объекта производного класса.

16. Абстрактный класс - это класс в котором

1) все компонентные функции "чисто виртуальные";
2) хотя бы одна компонентная функция "чисто виртуальная";
3) хотя бы одна компонентная функция не "чисто виртуальная";
4) все компонентные функции не "чисто виртуальные" .

17. В ООП инкапсуляция это

1) разделение элементов объекта, определяющих его устройство и поведение, служит для отделения интерфейса объекта и его реализации;

2) упрощенное описание системы, при котором выделяются только существенные детали;

3) разделение системы на части и дальнейшее исследование их в отдельности и взаимодействия между ними;

4) рассмотрение системы как иерархической структуры.

18. Формирование объекта класса по шаблону класса выполняется

1) самой программой на этапе выполнения при выделении памяти объекту;
2) компилятором на этапе компиляции при определении объекта;
3) программистом при редактировании исходного текста;
4) пользователя программы по запросу от нее.

19. Конструктор копирования класса вызывается

1) при выполнении операции присваивания;
2) при передаче объекта функции в качестве параметра по значению;
3) при копировании исходного файла программы;
4) при копировании объектного или исполняемого файлов.

20. Объектная декомпозиция это

1) разделение элементов объекта, определяющих его устройство и поведение, служит для отделения интерфейса объекта от его реализации;

2) упрощенное описание системы, при котором выделяются только существенные детали;

3) разделение системы на части и дальнейшее исследование их в отдельности и взаимодействия между ними;

4) рассмотрение системы как иерархической структуры.

21. Абстрагирование это

1) разделение элементов объекта, определяющих его устройство и поведение, служит для отделения интерфейса объекта от его реализации;

2) упрощенное описание системы, при котором выделяются только существенные детали;

3) разделение системы на части и дальнейшее исследование их в отдельности и взаимодействия между ними;

4) рассмотрение системы как иерархической структуры.

22. Компоненты классов C++, не доступные извне, описываются с использованием ключевого слова

1) public;
2) private;
3) protected;
4) friend.

21. Статические компоненты класса отличаются от обычных тем, что

1) разделяются всеми объектами класса;
2) имеют статическую продолжительность существования;
3) недоступны извне класса;
4) являются константами.

22. При определении компонентной функции внутри класса она рассматривается компилятором как

1) статическая;

2) виртуальная;
3) подставляемая;
4) защищенная.

23. Ключевое слово C++ this используется при реализации компонентных функций класса как
1) имя объекта класса, для которого вызвана функция;
2) имя класса, компонентом которого является функция;
3) имя указателя на данную компонентную функцию;
4) имя указателя на объект класса, для которого вызвана функция.

24. Конструктор класса предназначен для
1) инициализации компонентных данных объекта класса;
2) построения объекта класса на основании шаблона;
3) выделения памяти под статические компоненты класса;
4) присвоения имени объекту класса.

25. Конструктор класса не вызывается при
1) определении объекта класса в глобальной области программы;
2) размещении объекта в динамической памяти;
3) присвоении одному объекту класса значения другого;
4) передаче объекта функции как параметр по значению.

26. При использовании библиотеки потокового ввода вывода: #include <iostream.h> переменная cout описывает объект
1) стандартного потока вывода;
2) файлового потока вывода;
3) строкового потока вывода;
4) двунаправленного стандартного потока.

27. Конструктор копирования вызывается при
1) определении объекта класса в глобальной области программы;
2) размещении объекта в динамической памяти;
3) присвоении одному объекту класса значения другого;
4) передаче объекта функции как параметр по значению.

28. При использовании библиотеки потокового ввода вывода: #include <iostream.h> переменная cin описывает объект
1) стандартного потока ввода;
2) файлового потока ввода;
3) строкового потока ввода;
4) двунаправленного стандартного потока.

29. Какой тип имеет конструктор класса?
1) void;
2) char;
3) int;
4) не имеет типа.

30. Явным образом определенный деструктор класса может использоваться для
1) уничтожения статических компонентных данных;
2) освобождения динамической памяти, выделенной при инициализации объекта;

3) обнуления компонентных данных объекта;
4) изменения атрибутов объекта.

31.Какой из файлов заголовков необходимо подключить к программе, использующей ввод данных из файла с помощью средств библиотеки потокового ввода-вывода?

1) iostream.h;
2) strstream.h
3) fstream.h;
4) stdlib.h.

32. Деструктор не вызывается при

1) выходе из блока, в котором определен объект;
2) освобождении памяти, выделенной для динамического объекта;
3) выходе из функции для локального объекта функции;
4) применении к объекту оператора отрицания.

33. Дружественные функции имеют доступ

1) только к открытым компонентам класса;
2) только к закрытым компонентам класса;
3) ко всем компонентам класса;
4) только к компонентным данным.

34. Перегрузка стандартных операций языка C++ разрешена для

1) переменных стандартных типов данных;
2) переменных типов перечислений;
3) переменных-объектов классов;
4) переменных всех типов.

Раздел 4. Структуры данных и алгоритмы

1. Какая операция не характерна для очереди как структуры данных?

1) Вставка;
2) Сортировка;
3) Удаление;
4) Получение размера.

2. Стек это специальным образом организованная память, работающая по принципу

1) первый вошел - первый вышел;
2) первый вошел - последний вышел;
3) последний вошел - последний вышел;
4) вошел и не вышел.

3. Очередь – это специальным образом организованная память, работающая по принципу

1) первый вошел - первый вышел;
2) первый вошел - последний вышел;
3) последний вошел - первый вышел;
4) вошел и не вышел.

4. Очередь реализована с помощью массива. front и rear – индексы первого и последнего элементов очереди в массиве. MaxQSize – максимальное число элементов очереди (размер

массива). С применением какой операции происходит извлечение элемента из очереди?

1) front++;
2) rear++;
3) front = (front+1)%MaxQSize;
4) rear = (rear+1)%MaxQSize;

5. Очередь реализована с помощью массива. front и rear – индексы первого и последнего элементов очереди в массиве. MaxQSize – максимальное число элементов очереди (размер массива). С применением какой операции происходит вставка элемента в очередь?

1) front++;
2) rear++;
3) front = (front+1)%MaxQSize;
4) rear = (rear+1)%MaxQSize;

6. Какая из операций определяет специфику очереди приоритетов?

1) вставка данных;
2) извлечение данных;
3) проверка состояния очереди;
4) определение длины очереди.

7. Какая из структур данных обязательно требует перегрузки стандартного оператора для содержащихся в ней элементов – объектов некоторого класса?

1) стек;
2) очередь;
3) очередь приоритетов;
4) дерево общего вида.

8. Структура элемента связного списка обязательно имеет поле

1) указателя на следующий элемент;
2) указателя на первый элемент;
3) указателя на последний элемент;
4) указателя на массив со списком элементов.

9. Узлы бинарного дерева не содержат информации о

1) левом поддереве узла;
2) родителе узла;
3) правом поддереве узла;
4) данных узла.

10. Преимуществом связных списков перед массивами не является

1) быстрый доступ к элементу;
2) быстрая вставка элемента;
3) экономия памяти;
4) быстрое удаление элемента;

11. Сколько указателей нужно изменить при извлечении узла из середины двусвязного списка у оставшихся узлов?

1) 1;
2) 2;
3) 4;
4) ни одного.

12. Сколько указателей нужно настроить при вставке узла в середину двусвязного списка?

1) 1;
2) 2;
3) 4;
4) ни одного.

13. Функция DeleteNode извлекает узел с заданным содержимым из связного списка, размещенного в динамической памяти, и удаляет извлеченный узел. Вызывающей стороне функция должна сообщить об успехе своих действий. Какой прототип должна иметь функция (Node – имя класса узлов, type – тип поля данных узла)?

1) int DeleteNode(Node * & n, type& d);
2) void DeleteNode(Node * & n, type& d);
3) int DeleteNode(Node * n, type& d);
4) void DeleteNode(Node * n, type& d);

14. Прямой метод прохождения бинарного дерева предусматривает следующий порядок действий

1) прохождение левого поддерева – посещение узла – прохождение правого поддерева;
2) прохождение левого поддерева – прохождение правого поддерева – посещение узла;
3) посещение узла – прохождение левого поддерева – прохождение правого поддерева.

15. Обратный метод прохождения бинарного дерева предусматривает следующий порядок действий

1) прохождение левого поддерева – посещение узла – прохождение правого поддерева;
2) прохождение левого поддерева – прохождение правого поддерева – посещение узла;
3) посещение узла – прохождение левого поддерева – прохождение правого поддерева.

16. Симметричный метод прохождения бинарного дерева предусматривает следующий порядок действий

1) прохождение левого поддерева – посещение узла – прохождение правого поддерева;
2) прохождение левого поддерева – прохождение правого поддерева – посещение узла;
3) посещение узла – прохождение левого поддерева – прохождение правого поддерева.

17. Полное бинарное дерево глубины N содержит

1) N^2 узлов (^ – знак возведения в степень);
2) 2^N узлов;
3) 2^(N+1) узлов;
4) 2^(N+1) –1 узлов.

18. Какой порядок имеет алгоритм бинарного поиска в среднем случае?

1) n^2 (^ – знак возведения в степень);
2) n;
3) nlog(n) (логарифм по основанию 2);
4) 4) log(n).

19. Преимуществом последовательного поиска перед бинарным не является
1) возможность работы с неупорядоченными последовательностями;
2) возможность работы со связными списками;
3) большая вычислительная эффективность в среднем случае;
4) большая вычислительная эффективность в лучшем случае.

20. Какой порядок имеет сортировка вставками в среднем случае?
1) n^2;
2) n;
3) nlog(n) (логарифм по основанию 2);
4) 1/n.

21. Какой порядок имеет сортировка вставками в среднем случае?
1) n^2;
2) n;
3) nlog(n) (логарифм по основанию 2);
4) 1/n.

22. Какой порядок имеет "быстрая" сортировка Хоара в среднем случае?
1) n^2;
2) n;
3) nlog(n) (логарифм по основанию 2);
4) 1/n.

5. Интерактивные технологии и инновационные методы, используемые в образовательном процессе

В учебном процессе по дисциплине предусматриваются следующие интерактивные технологии и инновационные методы.

Лекционные занятия частично проводятся в форме компьютерных презентаций. Ряд презентаций содержит элементы анимации. Презентации ряду тем дополняются демонстрациями программ.

На практических занятиях проводятся дискуссии, разборы конкретных ситуаций, применяется метод мозгового штурма

В процедурах промежуточного и итогового контроля знаний используется компьютерное тестирование (см. п.4. «Контроль знаний»).