Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Новосибирский государственный технический университет»

Кафедра автоматизированных систем управления
Кафедра вычислительной техники
Кафедра автоматики

“УТВЕРЖДАЮ”
ДЕКАН АВТФ
к.т.н. Рева И. Л.
“___ ”_____ _____ г.

РАБОЧАЯ ПРОГРАММА УЧЕБНОЙ ДИСЦИПЛИНЫ
Информатика

Образовательная программа: 09.03.01 Информатика и вычислительная техника, профиль: Программное обеспечение компьютерных систем и сетей

Курс: 1, семестр : 1

Факультет автоматики и вычислительной техники,

		Семестр
№	**Вид деятельности**	**1**
1	Всего зачетных единиц (кредитов)	4
2	Всего часов	144
3	Всего занятий в контактной форме, час.	87
4	Лекции, час.	18
5	Практические занятия, час.	0
6	Лабораторные занятия, час.	54
7	из них в активной и интерактивной форме, час.	18
8	Аттестация, час.	2
9	Консультации, час.	13
10	Самостоятельная работа, час.	57
11	Виды самостоятельной работы (курсовой проект, курсовая работа, РГЗ, подготовка к контрольной работе)	РГЗ
12	Вид аттестации	Э

Новосибирск 2017

Рабочая программа составлена на основании федерального государственного образовательного стандарта высшего образования по направлению (специальности): 09.03.01 Информатика и вычислительная техника

ФГОС введен в действие приказом №5 от 12.01.2016 г. , дата утверждения: 09.02.2016 г.

Место дисциплины в структуре учебного плана: Блок 1, базовая

Рабочая программа разработана на основе компетентностной модели выпускника по направлению (специальности): 09.03.01 Информатика и вычислительная техника

Рабочая программа обсуждена на заседании кафедры

АСУ, протокол заседания кафедры №7 от 20.06.2017
ВТ, протокол заседания кафедры №6 от 20.06.2017
АВТ, протокол заседания кафедры №10/1 от 20.06.2017

Утверждена на совете факультета автоматики и вычислительной техники, протокол № 6 от 21.06.2017

Программу разработал:

старший преподаватель, Лауферман О. В.

Заведующий кафедрой:

профессор, д.т.н. Гриф М. Г.
доцент, к.т.н. Якименко А. А.
доцент, д.т.н. Жмудь В. А.

Ответственный за образовательную программу:

доцент Романов Е. Л.

1. Внешние требования

Таблица 1.1

***Компетенция ФГОС*: ОПК.5 способность решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности; *в части следующих результатов обучения*:**
з11. знать методы и средства проектирования программного обеспечения
з5. знать правовые основы информационной безопасности и принципы защиты авторского права на программные продукты
з6. знать сущность и значение информации в развитии современного общества, опасности и угроз, возникающие в этом процессе
у10. уметь использовать специализированные программные средства при решении профессиональных задач
у11. уметь использовать элементарные навыки алгоритмизации и программирования на одном из языков высокого уровня как средство программного моделирования изучаемых объектов и процессов
у12. уметь оценивать состояние и тенденции развития информационных технологий и информатики в современном обществе
у13. уметь проводить библиографическую и информационно-поисковую работы, использовать ее результаты при решении профессиональных задач и оформлении научных трудов
у3. уметь пользоваться наиболее распространенными офисными и математическими пакетами прикладных программ
у4. уметь осуществлять поиск информации в локальных и глобальных сетях
у5. уметь применять основные методы, способы и средства получения, хранения и переработки информации с помощью компьютеров и компьютерных средств
у8. владеть персональным компьютером как средством управления информацией

2. Требования НГТУ к результатам освоения дисциплины

Таблица 2.1

Результаты изучения дисциплины по уровням освоения (иметь представление, знать, уметь, владеть)	**Формы организации занятий**
Информатика	
ОПК.5.з5 знать правовые основы информационной безопасности и принципы защиты авторского права на программные продукты	
1.об информации, методах ее хранения, обработки и передачи	Лекции; Лабораторные работы; Самостоятельная работа
ОПК.5.з6 знать сущность и значение информации в развитии современного общества, опасности и угроз, возникающие в этом процессе	
2.об архитектуре компьютера с точки зрения реализации алгоритма и программы	Лекции; Лабораторные работы; Самостоятельная работа
ОПК.5.з11 знать методы и средства проектирования программного обеспечения	
3.о современных алгоритмических языках, их области применения и особенностях	Лекции; Лабораторные работы; Самостоятельная работа
4.о структуре языков программирования - уровнях описания данных, операций и выражений, операторов, модулей	Лекции; Лабораторные работы; Самостоятельная работа
ОПК.5.у4 уметь осуществлять поиск информации в локальных и глобальных сетях	
5. о средствах определения данных (типы данных, переменные), принятых в большинстве языков программирования	Лекции; Лабораторные работы; Самостоятельная работа
ОПК.5.з11 знать методы и средства проектирования программного обеспечения	
6.о технологии проектирования сложных модульных программ	Лекции; Лабораторные работы; Самостоятельная работа
ОПК.5.у5 уметь применять основные методы, способы и средства получения, хранения и переработки информации с помощью компьютеров и компьютерных средств	

7.о многообразии структур данных и их использовании для хранения, поиска и упорядочения данных	Лекции; Лабораторные работы; Самостоятельная работа
ОПК.5.у13 уметь проводить библиографическую и информационно-поисковую работы, использовать ее результаты при решении профессиональных задач и оформлении научных трудов	
8.базовые понятия информатики и вычислительной техники	Лекции; Самостоятельная работа
ОПК.5.з6 знать сущность и значение информации в развитии современного общества, опасности и угроз, возникающие в этом процессе	
9.предмет и основные методы информатики	Лекции; Самостоятельная работа
ОПК.5.у11 уметь использовать элементарные навыки алгоритмизации и программирования на одном из языков высокого уровня как средство программного моделирования изучаемых объектов и процессов	
10.формы представления числовой и символьной информации	Лекции; Лабораторные работы; Самостоятельная работа
ОПК.5.з11 знать методы и средства проектирования программного обеспечения	
11.способы определения переменных базовых типов данных, операций, операторов и функций в языке Си	Лекции; Лабораторные работы; Самостоятельная работа
12.алгоритмы поиска и сортировки данных	Лекции; Лабораторные работы; Самостоятельная работа
14.свойства и принципы работы с производными типами данных - указателями, массивами, структурами, функциями	Лекции; Лабораторные работы; Самостоятельная работа
15.основы технологии структурного программирования	Лекции; Лабораторные работы; Самостоятельная работа
ОПК.5.у3 уметь пользоваться наиболее распространенными офисными и математическими пакетами прикладных программ	
16.анализировать существующие и разрабатывать собственные программы с использованием стандартных фрагментов алгоритмов	Лабораторные работы; Самостоятельная работа
ОПК.5.у8 владеть персональным компьютером как средством управления информацией	
17.кодировать, транслировать и отлаживать программы на языке С/ С++	Лабораторные работы; Самостоятельная работа
ОПК.5.у3 уметь пользоваться наиболее распространенными офисными и математическими пакетами прикладных программ	
18.использовать технологию пошагового модульного проектирования программ	Лекции; Лабораторные работы; Самостоятельная работа
ОПК.5.у10 уметь использовать специализированные программные средства при решении профессиональных задач	
19.проектировать программы обработки символьной информации	Лекции; Лабораторные работы; Самостоятельная работа
ОПК.5.у11 уметь использовать элементарные навыки алгоритмизации и программирования на одном из языков высокого уровня как средство программного моделирования изучаемых объектов и процессов	
20.использовать динамическую память при обработке данных заранее неизвестного объема и размерности	Лекции; Лабораторные работы; Самостоятельная работа
21.проектироваать сложные модульные программы	Лекции; Лабораторные работы; Самостоятельная работа
ОПК.5.у12 уметь оценивать состояние и тенденции развития информационных технологий и информатики в современном обществе	
22.уметь своевременно обращаться к соответствующей инструментальной технологии (информационной)	Лекции; Лабораторные работы; Самостоятельная работа

3. Содержание и структура учебной дисциплины

Таблица 3.1

Темы лекций	Активные формы, час.	Часы	Ссылки на результаты обучения
Семестр: 1			
Дидактическая единица: Основные понятия и определения			
1. Понятие информатики; понятие информации и ее измерение; количество и качество информации; информационный процесс в автоматизированных системах. Предмет и задачи информатики. Информация и информатика. Структура курса. Его связь с другими дисциплинами учебного плана. Особенности предмета курса.	0	1	1, 10, 2, 22, 8, 9
2. Основные устройства и ресурсы ЭВМ. Архитектура ЭВМ. Организация компьютера: устройства ввода-вывода, память (доступ к памяти, ячейка, адресация, содержимое ячейки, запись, чтение). Режимы адресации. Носители информации и технические средства для хранения данных.	0	1	1, 10, 2, 22, 8
Дидактическая единица: Структурное программирование			
3. Структура программы и классы памяти. Объекты и их атрибуты. Директивы препроцессора.	0	2	1, 10, 11, 20, 22, 5, 7, 8
4. Программа на языке высокого уровня. Структура программы на языке Си. Лексемы языка. Основные операторы языка.	0	2	15, 18, 2, 3, 4, 6, 8, 9
5. Процедуры и функции. Функции. Определение, объявление и вызов функций в языке Си. Параметры функций. Способы передачи параметров. Параметры функции main(). Массивы и функции.	0	2	12, 15, 18, 21, 3, 4, 6
Дидактическая единица: Алгоритмы и структуры данных.			
2. Стандартные типы данных. Переменные, адреса, указатели. Адресная арифметика. Массивы и указатели. Ссылки.	0	4	1, 10, 11, 14, 20, 4, 5, 7
6. Типы и структуры данных. Массивы: утверждения о массивах; записи. Методы организации данных. Типы данных: скаляры и структуры (массив, запись, структура). Уровни организации данных.	0	2	1, 12, 21, 4, 5, 7, 8, 9
7. Методы построения алгоритмов. Алгоритмы поиска и сортировки. Оценка эффективности алгоритмов.	0	2	12, 15, 21, 8
9. Алгоритмы. Типы алгоритмов. Способы реализации: итерация и рекурсия, параллельная обработка, сопрограммы. Свойства рекурсивных алгоритмов.	0	2	1, 12, 19, 2, 21, 22, 3

Таблица 3.2

Темы лабораторных работ	Активные формы, час.	Часы	Ссылки на результаты обучения	Учебная деятельность
Семестр: 1				
Дидактическая единица: Структурное программирование				
1. Условные операторы. Операторы цикла.	4	8	10, 11, 15, 17, 18, 3, 5, 6	Обучающийся формулирует постановку задачи, определяет тестовые данные, выбирает алгоритм и типы переменных, кодирует, отлаживает и тестирует программу. Выбирает подходящую циклическую конструкцию
5. Функции. Способы передачи параметров	4	10	11, 15, 16, 17, 18, 21, 22, 3, 4, 6	Обучающийся разбивает задачу на подзадачи, определяет сигнатуры и содержание функций, организует вызов функций в программе, использует в качестве параметров, передаваемых в функции: константы, переменные, выражения
Дидактическая единица: Алгоритмы и структуры данных.				
2. Массивы целых чисел	2	8	1, 10, 15, 16, 17, 18, 2, 22, 3, 5, 7	Обучающийся выбирает тип и размер массива, набор операций над его элементами, наглядно представляет результаты на экране
3. Символьные массивы - строки	2	8	1, 10, 11, 16, 17, 19, 20, 3, 5, 7	Обучающийся использует библиотечные функции для обработки отдельных символов и строк. Выполняет действия над строками без использования библиотек
4. Методы сортировки	4	8	10, 12, 15, 18, 19, 22, 3, 4	Обучающийся иследует методы сортировки данных, кодирует один из них, отображает "по шагам" процесс сортировки на экране, оценивает эффективность данного алгоритма
6. Массивы указателей на строки.	2	12	12, 14, 17, 18, 19, 20, 21, 22	Обучающийся использует динамическое распределение памяти, объединяет указатели в структуры, применяет операции адресной арифметики, определяет преимущества динамических объектов над статическими структурами

Таблица 3.3

Темы для самостоятельного изучения	Активные формы, час.	Часы	Ссылки на результаты обучения	Учебная деятельность
Семестр: 1				
Дидактическая единица: Основные понятия и определения				
2. Информационные технологии	0	5	1, 15, 2, 22, 5, 6, 8, 9	темы:"информационный ресурс и его составляющие", "информационные технологии", "технические и программные средства информационных технологий", "основные виды обработки данных", "обработка аналоговой и цифровой информации"; "устройства обработки данных и их характеристики", "понятие и свойства алгоритма", "принципы программного управления", "функциональная и структурная организация компьютера", "сетевые технологии обработки
Дидактическая единица: Структурное программирование				
1. Проектирование алгоритмов и структур данных. Оценка эффективности	0	5	1, 10, 12, 15, 16, 17, 18, 21, 3, 4, 7	Обучающийся готовится к лекционным и лабораторным занятиям: самостоятельно работает с основной и дополнительной литературой по данной дисциплине, развивает навыки программирования
Дидактическая единица: Алгоритмы и структуры данных.				
3. Информация на внешних носителях	0	10	1, 10, 12, 16, 19, 22, 5, 7, 8	Обучающийся изучает темы: "типы и структуры данных", "организация данных на устройствах с прямым и последовательным доступом", "файлы данных", "файловые структуры", "носители информации и технические средства для хранения данных"

4. Самостоятельная работа обучающегося

№	Виды самостоятельной работы	Ссылки на результаты обучения	Часы на выполнение	Часы на консультации
Семестр: 1				
1	РГЗ	10, 12, 16, 18, 2, 21, 22, 4, 6, 8	20	4

Обучающийся изучает темы: "Структуры в языке С", "Файлы", "Структуры данных в бинарных файлах","Сортировка и поиск данных", подробная информация приведена в приложениях №2 №5 : Лауферман О. В. Информатика. Ч. 2 : учебное пособие / О. В. Лауферман, Р. Г. Шахмаметов ; Новосиб. гос. техн. ун-т. - Новосибирск, 2008. - 74, [1] с.. - Режим доступа: http://elibrary.nstu.ru/source?bib_id=vtls000082408				
2	Подготовка к лабораторным работам	1, 11, 14, 16, 18, 20, 22, 3, 5, 7, 9	5	2
Обучающийся работает с рекомендованной литературой, готовится к промежуточному контролю: повторение конспектов лекций, ответы на "вопросы на полях лекционных занятий", подробная информация приведена в приложениях №1 №4 №5 №6 : Лауферман О. В. Информатика. Ч. 2 : учебное пособие / О. В. Лауферман, Р. Г. Шахмаметов ; Новосиб. гос. техн. ун-т. - Новосибирск, 2008. - 74, [1] с.. - Режим доступа: http://elibrary.nstu.ru/source?bib_id=vtls000082408 Лауферман О. В. Информатика [Электронный ресурс] : электронный учебно-методический комплекс [для групп АВТ-х12, АВТ-х13] / О. В. Лауферман ; Новосиб. гос. техн. ун-т. - Новосибирск, [2016]. - Режим доступа: http://elibrary.nstu.ru/source?bib_id=vtls000232487. - Загл. с экрана.				
3	Дополнительная учебная деятельность	10, 12, 15, 17, 19, 2, 21, 4, 6, 8	5	2
Обучающийся изучает темы: "Указатели на функции", "Функции с переменным количеством параметров", "Перегрузка функций", подробная информация приведена в приложениях №1 №2 №5 : Лауферман О. В. Информатика. Ч. 2 : учебное пособие / О. В. Лауферман, Р. Г. Шахмаметов ; Новосиб. гос. техн. ун-т. - Новосибирск, 2008. - 74, [1] с.. - Режим доступа: http://elibrary.nstu.ru/source?bib_id=vtls000082408				
4	Тест №1 на тему "Структурное программирование", тест №2 "Алгоритмы и структуры данных", защита РГЗ, подготовка к экзамену	1, 11, 14, 16, 18, 20, 22, 3, 5, 7, 9	7	2
Обучающийся работает с конспектами лекций, отвечает на "вопросы для самопроверки знаний", подробная информация приведена в приложениях №1 №2 №5 : Лауферман О. В. Информатика. Ч. 2 : учебное пособие / О. В. Лауферман, Р. Г. Шахмаметов ; Новосиб. гос. техн. ун-т. - Новосибирск, 2008. - 74, [1] с.. - Режим доступа: http://elibrary.nstu.ru/source?bib_id=vtls000082408				
5	Самостоятельное изучение теоретического материала	1, 10, 12, 15, 16, 17, 18, 19, 2, 21, 22, 3, 4, 5, 6, 7, 8, 9	23	3
Студент изучает темы, приведенные в таблице 3.3 , подробная информация приведена в приложениях №4 №5 №6 : Лауферман О. В. Информатика. Ч. 2 : учебное пособие / О. В. Лауферман, Р. Г. Шахмаметов ; Новосиб. гос. техн. ун-т. - Новосибирск, 2008. - 74, [1] с.. - Режим доступа: http://elibrary.nstu.ru/source?bib_id=vtls000082408				

5. Технология обучения

Для организации и контроля самостоятельной работы обучающихся, а также проведения консультаций применяются информационно-коммуникационные технологии (табл. 5.1).

Таблица 5.1

Деятельность	Информационно-коммуникационные технологии
Информирование	e-mail; Портал НГТУ
Консультирование	e-mail; Чат
Контроль	Среда электронного обучения НГТУ; ЭБС
Размещение учебных материалов	Личный типовой сайт; Социальные сети

Таблица 5.2

Активные и интерактивные формы проведения занятий

№	Наименование активных форм
1	Лекция в форме дискуссии
Краткое описание применения:	
Подробная информация об использовании технологии приводится в "Лауферман О. В. Информатика. Ч. 2 : учебное пособие / О. В. Лауферман, Р. Г. Шахмаметов ; Новосиб. гос. техн. ун-т. - Новосибирск, 2008. - 74, [1] с.. - Режим доступа: http://elibrary.nstu.ru/source?bib_id=vtls000082408"	

6. Правила аттестации обучающихся по учебной дисциплине

Для аттестации обучающихся по дисциплине используется балльно-рейтинговая система (БРС), позволяющая выставлять оценки по традиционной шкале и 15-уровневой ECTS. Краткая информация о БРС приведена в табл. 6.1.

Таблица 6.1

Оцениваемые виды деятельности обучающихся	Мин. балл	Максимальный балл
Семестр: 1		
Лабораторная №1: Условные операторы. Операторы цикла	5	8
Лабораторная №2: Массивы целых чисел	5	8
Лабораторная №3: Символьные массивы	5	8
Лабораторная №4: Методы сортировки	5	8
Лабораторная №5: Функции. Способы передачи параметров	5	8
Лабораторная №6: Массивы указателей на строки	8	10
РГЗ: Структуры данных	4	10
Экзамен:	0	40

В таблице 6.2 представлено соответствие форм контроля заявляемым требованиям к результатам освоения дисциплины.

Таблица 6.2

Коды компетенций ФГОС	Результаты обучения	Формы контроля		
		Защита Л/Р	Защита РГЗ	Экзамен
ОПК.5	з11. знать методы и средства проектирования программного обеспечения	+	+	+
	з5. знать правовые основы информационной безопасности и принципы защиты авторского права на программные продукты		+	+
	з6. знать сущность и значение информации в развитии современного общества, опасности и угроз, возникающие в этом процессе		+	+
	у10. уметь использовать специализированные программные средства при решении профессиональных задач	+		+
	у11. уметь использовать элементарные навыки алгоритмизации и программирования на одном из языков высокого уровня как средство программного моделирования изучаемых объектов и процессов	+	+	+

	у12. уметь оценивать состояние и тенденции развития информационных технологий и информатики в современном обществе	+		+
	у13. уметь проводить библиографическую и информационно-поисковую работы, использовать ее результаты при решении профессиональных задач и оформлении научных трудов		+	+
	у3. уметь пользоваться наиболее распространенными офисными и математическими пакетами прикладных программ	+	+	+
	у4. уметь осуществлять поиск информации в локальных и глобальных сетях		+	+
	у5. уметь применять основные методы, способы и средства получения, хранения и переработки информации с помощью компьютеров и компьютерных средств		+	+
	у8. владеть персональным компьютером как средством управления информацией	+		+

Фонд оценочных средств по дисциплине представлен в приложении № 1 к рабочей программе.

7. Литература

Основная литература

1. Лыгина Н. И. Информатика : учебное пособие / Н. И. Лыгина, О. В. Лауферман; Новосиб. гос. техн. ун-т. - Новосибирск, 2017 - Режим доступа:http://elibrary.nstu.ru/source?bib_id=vtls000234957

2. Романов Е. Л. Си/Си ++. От дилетанта до профессионала [Электронный ресурс] : электронное учебное пособие : для 1-2 курсов направления 230100 "Информатика и вычислительная техника / Романов Е. Л. - Новосибирск, 2010. - 1 электрон. опт. диск (CD-ROM). - Режим доступа: http://elibrary.nstu.ru/source?bib_id=vtls000134024. - Загл. с этикетки диска.- Рег. свидетельство №18891.

3. Никлаус Вирт Алгоритмы и структуры данных [Электронный ресурс]/ Вирт Никлаус— Электрон. текстовые данные.— Саратов: Профобразование, 2017.— 272 c.— Режим доступа: http://www.iprbookshop.ru/63821.html.— ЭБС «IPRbooks»

Дополнительная литература

1. Советов, Б.Я. Информационные технологии: теоретические основы. [Электронный ресурс] / Б.Я. Советов, В.В. Цехановский. — Электрон. дан. — СПб. : Лань, 2017. — 444 с. — Режим доступа: http://e.lanbook.com/book/93007 — Загл. с экрана.

2. Кудинов, Ю.И. Основы современной информатики. [Электронный ресурс] / Ю.И. Кудинов, Ф.Ф. Пащенко. — Электрон. дан. — СПб. : Лань, 2017. — 256 с. — Режим доступа: http://e.lanbook.com/book/91902 — Загл. с экрана.

3. Кудинов, Ю.И. Практикум по основам современной информатики. [Электронный ресурс] / Ю.И. Кудинов, Ф.Ф. Пащенко, А.Ю. Келина. — Электрон. дан. — СПб. : Лань, 2011. — 352 с. — Режим доступа: http://e.lanbook.com/book/68471 — Загл. с экрана.

4. Кнут Д. Э. Искусство программирования. Т. 1, вып. 1 / Дональд Э. Кнут. - М. [и др.], 2007. - 150 с.

5. Кнут Д. Э. Искусство программирования. Т. 2 : пер. с англ. / Дональд Э. Кнут ; под общ. ред. Ю. В. Козаченко. - М. [и др.], 2007. - 828 с. : ил.

6. Кнут Д. Э. Искусство программирования. Т. 3 : пер. с англ. / Дональд Э. Кнут ; под общ. ред. Ю. В. Козаченко. - М. [и др.], 2007. - 822 с. : ил.

7. Дейтел Х. М. Как программировать на C++ / Х. М. Дейтел, П. Дж. Дейтел ; пер. с англ. под ред. В. В. Тимофеева. - М., 2007. - 799 с. : ил.

8. Павловская Т. А. Паскаль. Программирование на языке высокого уровня : [учебник по направлению "Информатика и вычислительная техника"] / Т. А. Павловская. - СПб. [и др.], 2010. - 460 с.

9. Крылов Е. В. Техника разработки программ. В 2 кн.. Кн. 1 : [учебник для вузов по направлениям "Информатика и вычислительная техника" и "Техника и технологии"] / Е. В. Крылов, В. А. Острейковский, Н. Г. Типикин. - М., 2007. - 374, [1] с. : ил.

Интернет-ресурсы

1. ЭБС НГТУ : http://elibrary.nstu.ru/

2. ЭБС «Издательство Лань» : https://e.lanbook.com/

3. ЭБС IPRbooks : http://www.iprbookshop.ru/

4. ЭБС "Znanium.com" : http://znanium.com/

5. :

8. Методическое и программное обеспечение

8.1 Методическое обеспечение

1. Лауферман О. В. Информатика [Электронный ресурс] : электронный учебно-методический комплекс [для групп АВТ-х12, АВТ-х13] / О. В. Лауферман ; Новосиб. гос. техн. ун-т. - Новосибирск, [2016]. - Режим доступа: http://elibrary.nstu.ru/source?bib_id=vtls000232487. - Загл. с экрана.

2. Лауферман О. В. Информатика. Ч. 2 : учебное пособие / О. В. Лауферман, Р. Г. Шахмаметов ; Новосиб. гос. техн. ун-т. - Новосибирск, 2008. - 74, [1] с.. - Режим доступа: http://elibrary.nstu.ru/source?bib_id=vtls000082408

8.2 Специализированное программное обеспечение

1 Microsoft Windows

2 Microsoft Office

9. Материально-техническое обеспечение

Компьютерный класс

№	Наименование	Назначение
1	Компьютер мультимедиа	Для проведения занятий с о студентами ИСР

Комплект оборудования

№	Наименование	Назначение
1	Проектор	Для учебных целей. Проведение лабораторных работ и практических занятийс показом слайдов, презентаций

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Новосибирский государственный технический университет»

Кафедра автоматизированных систем управления
Кафедра автоматики
Кафедра вычислительной техники

“УТВЕРЖДАЮ”
ДЕКАН АВТФ
к.т.н., доцент И.Л. Рева
“___ ”_____ _____ г.

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ

УЧЕБНОЙ ДИСЦИПЛИНЫ

Информатика

Образовательная программа: 09.03.01 Информатика и вычислительная техника, профиль: Программное обеспечение компьютерных систем и сетей

Новосибирск 2017

1. **Обобщенная структура фонда оценочных средств учебной дисциплины**

Обобщенная структура фонда оценочных средств по **дисциплине** Информатика приведена в Таблице 1.

Таблица 1

Формируемые компетенции	**Показатели сформированности компетенций (знания, умения, навыки)**	**Темы**	**Этапы оценки компетенций**	
			Мероприятия текущего контроля (курсовой проект, РГЗ(Р) и др.)	**Промежуточная аттестация (экзамен, зачет)**
ОПК.5 способность решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности	з5. знать правовые основы информационной безопасности и принципы защиты авторского права на программные продукты	Информационные технологии Информация на внешних носителях Основные устройства и ресурсы ЭВМ. Архитектура ЭВМ. Организация компьютера: устройства ввода-вывода, память (доступ к памяти, ячейка, адресация, содержимое ячейки, запись, чтение). Режимы адресации. Носители информации и технические средства для хранения данных. Понятие информатики; понятие информации и ее измерение; количество и качество информации; информационный процесс в автоматизированных системах. Предмет и задачи информатики. Информация и информатика. Структура курса. Его связь с другими дисциплинами учебного плана. Особенности предмета курса.	РГЗ, разделы №1-8	Экзамен, вопросы №1-25
	з6. знать сущность и значение информации в развитии современного общества, опасности и угроз, возникающие в этом процессе	Алгоритмы. Типы алгоритмов. Способы реализации: итерация и рекурсия, параллельная обработка, сопрограммы. Свойства рекурсивных алгоритмов. Информационные технологии Основные устройства и ресурсы ЭВМ. Архитектура ЭВМ. Организация компьютера: устройства ввода-вывода, память (доступ к памяти, ячейка, адресация, содержимое ячейки, запись, чтение). Режимы адресации. Носители информации и технические средства для хранения данных. Понятие информатики; понятие информации и ее измерение; количество и качество информации; информационный процесс в автоматизированных системах. Предмет и задачи информатики. Информация и информатика. Структура курса. Его связь с другими дисциплинами учебного плана. Особенности предмета курса.	РГЗ, разделы №1-8	Экзамен, вопросы №1-25
ОПК.5	з11. знать методы и средства проектирования программного обеспечения	Алгоритмы. Типы алгоритмов. Способы реализации: итерация и рекурсия, параллельная обработка, сопрограммы. Свойства рекурсивных алгоритмов. Информационные технологии Массивы указателей на строки. Массивы целых чисел Методы построения алгоритмов. Алгоритмы поиска и сортировки. Оценка эффективности алгоритмов. Методы сортировки Программа на языке высокого уровня. Структура программы на языке Си. Лексемы языка. Основные операторы языка. Проектирование алгоритмов и структур данных. Оценка эффективности Процедуры и функции. Функции. Определение, объявление и вызов функций в языке Си. Параметры функций. Способы передачи параметров. Параметры функции main(). Массивы и функции. Стандартные типы данных. Переменные, адреса, указатели. Адресная арифметика. Массивы и указатели. Ссылки. Структура программы и классы памяти. Объекты и их атрибуты. Директивы препроцессора. Типы и структуры данных. Массивы: утверждения о массивах; записи. Методы организации данных. Типы данных: скаляры и структуры (массив, запись, структура). Уровни организации данных. Условные операторы. Операторы цикла. Функции. Способы передачи параметров	Отчет по лабораторной работе №1-8, РГЗ, разделы№1-8	Экзамен, вопросы №1-25

ОПК.5	у3. уметь пользоваться наиболее распространенными офисными и математическими пакетами прикладных программ	Массивы целых чисел Методы сортировки Программа на языке высокого уровня. Структура программы на языке Си. Лексемы языка. Основные операторы языка. Проектирование алгоритмов и структур данных. Оценка эффективности Процедуры и функции. Функции. Определение, объявление и вызов функций в языке Си. Параметры функций. Способы передачи параметров. Параметры функции main(). Массивы и функции. Символьные массивы - строки Функции. Способы передачи параметров	Отчет по лабораторной работе №1-8, РГЗ, разделы№1-8	Экзамен, вопросы №1-25
ОПК.5	у4. уметь осуществлять поиск информации в локальных и глобальных сетях	Массивы целых чисел Символьные массивы - строки Стандартные типы данных. Переменные, адреса, указатели. Адресная арифметика. Массивы и указатели. Ссылки. Структура программы и классы памяти. Объекты и их атрибуты. Директивы препроцессора. Типы и структуры данных. Массивы: утверждения о массивах; записи. Методы организации данных. Типы данных: скаляры и структуры (массив, запись, структура). Уровни организации данных.	РГЗ, разделы№1-8	Экзамен, вопросы №1-25
ОПК.5	у5. уметь применять основные методы, способы и средства получения, хранения и переработки информации с помощью компьютеров и компьютерных средств	Информация на внешних носителях Проектирование алгоритмов и структур данных. Оценка эффективности Стандартные типы данных. Переменные, адреса, указатели. Адресная арифметика. Массивы и указатели. Ссылки. Типы и структуры данных. Массивы: утверждения о массивах; записи. Методы организации данных. Типы данных: скаляры и структуры (массив, запись, структура). Уровни организации данных.	РГЗ, разделы№1-8	Экзамен, вопросы №1-25
ОПК.5	у8. владеть персональным компьютером как средством управления информацией	Массивы указателей на строки. Массивы целых чисел Проектирование алгоритмов и структур данных. Оценка эффективности Символьные массивы - строки Условные операторы. Операторы цикла. Функции. Способы передачи параметров	Отчет по лабораторной работе №1-8	Экзамен, вопросы №1-25
ОПК.5	у10. уметь использовать специализированные программные средства при решении профессиональных задач	Алгоритмы. Типы алгоритмов. Способы реализации: итерация и рекурсия, параллельная обработка, сопрограммы. Свойства рекурсивных алгоритмов. Массивы указателей на строки. Методы сортировки Символьные массивы - строки	Отчет по лабораторной работе №1-8	Экзамен, вопросы №1-25
ОПК.5	у11. уметь использовать элементарные навыки алгоритмизации и программирования на одном из языков высокого уровня как средство программного моделирования изучаемых объектов и процессов	Алгоритмы. Типы алгоритмов. Способы реализации: итерация и рекурсия, параллельная обработка, сопрограммы. Свойства рекурсивных алгоритмов. Массивы указателей на строки. Массивы целых чисел Методы построения алгоритмов. Алгоритмы поиска и сортировки. Оценка эффективности алгоритмов. Проектирование алгоритмов и структур данных. Оценка эффективности Процедуры и функции. Функции. Определение, объявление и вызов функций в языке Си. Параметры функций. Способы передачи параметров. Параметры функции main(). Массивы и функции. Символьные массивы - строки Стандартные типы данных. Переменные, адреса, указатели. Адресная арифметика. Массивы и указатели. Ссылки. Структура программы и классы памяти. Объекты и их атрибуты. Директивы препроцессора. Функции. Способы передачи параметров	Отчет по лабораторной работе №1-8, РГЗ, разделы№1-8	Экзамен, вопросы №1-25
ОПК.5	у12. уметь оценивать состояние и тенденции	Информационные технологии Массивы указателей на строки. Массивы целых чисел Методы сортировки Функции. Способы передачи параметров	Отчет по лабораторной работе №1-8	Экзамен, вопросы №1-25

	развития информационны х технологий и информатики в современном обществе			
ОПК.5	у13. уметь проводить библиографичес кую и информационно-поисковую работы, использовать ее результаты при решении профессиональн ых задач и оформлении научных трудов	Информационные технологии Информация на внешних носителях Основные устройства и ресурсы ЭВМ. Архитектура ЭВМ. Организация компьютера: устройства ввода-вывода, память (доступ к памяти, ячейка, адресация, содержимое ячейки, запись, чтение). Режимы адресации. Носители информации и технические средства для хранения данных. Понятие информатики; понятие информации и ее измерение; количество и качество информации; информационный процесс в автоматизированных системах. Предмет и задачи информатики. Информация и информатика. Структура курса. Его связь с другими дисциплинами учебного плана. Особенности предмета курса.	РГЗ, разделы№1-8	Экзамен, вопросы №1-25

2. Методика оценки этапов формирования компетенций в рамках дисциплины.

Промежуточная аттестация по дисциплине проводится в 1 семестре - в форме экзамена, который направлен на оценку сформированности компетенций ОПК.5.

Экзамен проводится письменно в соответствии с требованиями, изложенными в паспорте экзамена.

Кроме того, сформированность компетенции проверяется при проведении мероприятий текущего контроля, указанных в таблице 1 раздела 1.

В 1 семестре обязательным этапом текущей аттестации является расчетно-графическое задание (РГЗ). Требования к выполнению РГЗ, состав и правила оценки сформулированы в паспорте РГЗ.

Общие правила выставления оценки по дисциплине определяются балльно-рейтинговой системой, приведенной в рабочей программе дисциплины.

На основании приведенных далее критериев можно сделать общий вывод о сформированности компетенции ОПК.5, за которые отвечает дисциплина, на разных уровнях.

Общая характеристика уровней освоения компетенций.

Ниже порогового. Уровень выполнения работ не отвечает большинству основных требований, теоретическое содержание курса освоено частично, пробелы могут носить существенный характер, необходимые практические навыки работы с освоенным материалом сформированы не достаточно, большинство предусмотренных программой обучения учебных заданий не выполнены или выполнены с существенными ошибками.

Пороговый. Уровень выполнения работ отвечает большинству основных требований, теоретическое содержание курса освоено частично, но пробелы не носят существенного характера, необходимые практические навыки работы с освоенным материалом в основном сформированы, большинство предусмотренных программой обучения учебных заданий выполнено, некоторые виды заданий выполнены с ошибками.

Базовый. Уровень выполнения работ отвечает всем основным требованиям, теоретическое содержание курса освоено полностью, без пробелов, некоторые практические навыки работы с освоенным материалом сформированы недостаточно, все предусмотренные программой обучения учебные задания выполнены, качество выполнения ни одного из них не оценено минимальным числом баллов, некоторые из выполненных заданий, возможно, содержат ошибки.

Продвинутый. Уровень выполнения работ отвечает всем требованиям, теоретическое содержание курса освоено полностью, без пробелов, необходимые практические навыки работы с освоенным материалом сформированы, все предусмотренные программой обучения учебные задания выполнены, качество их выполнения оценено числом баллов, близким к максимальному.

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Новосибирский государственный технический университет»
Кафедра автоматизированных систем управления
Кафедра автоматики
Кафедра вычислительной техники

Паспорт экзамена

по дисциплине «Информатика», 1 семестр

1. Методика оценки

Экзамен проводится в письменной форме, по билетам.
Билет формируется по следующему правилу:

1. вопрос выбирается из диапазона вопросов 1-25 (п. 4);
2. задача из таблицы 1 (уровень сложности 1);
3. задача из таблицы 1 (уровень сложности 2 или 3).

В ходе экзамена преподаватель вправе задавать студенту дополнительные вопросы из общего перечня (п. 4).

Таблица 1

Уровень сложности/ оценка	Варианты заданий
1/ «удовлетво-рительно»	1. Дан массив целых чисел A. Записать в массив B сначала все четные числа из массива A, затем все нечетные, в конце массива разместить нулевые элементы. Вывести массив B 2. Вычислить сумму и количество элементов массива A, стоящих на нечетных позициях, умножить все элементы на четных позициях на число M, вывести новый массив 3. Дан массив целых чисел A. Записать в массив B сначала все положительные числа из массива A, затем все отрицательные, в конце массива разместить нулевые элементы. Вывести массив B 4. Дан упорядоченный по возрастанию массив A, удалить из него K-ый элементу, и вставить заданное число M так, чтобы не нарушилась последовательность 5. Дан массив целых чисел A, состоящий из чисел кратных 3 и 5. Записать в массив B сначала все числа, кратные только 3, затем все числа, кратные только 5, в конце массива разместить числа, кратные и 3 и 5. Вывести массив B 6. Вывести индекс элемента, стоящего посередине между максимальным и минимальным элементами массива A, вычислить полусумму этих элементов 7. Дан массив целых чисел A. Записать в массив B сначала все числа, которые являются целыми квадратами (например, 25, 64, 49) а, затем все остальные элементы массива A. Вывести массив B 8. Вычесть из каждого элемента массива A значение минимального элемента, исключить из нового массива нулевые элементы, вывести новый массив и его размерность 9. Дан массив целых чисел A. Записать в массив B сначала все элементы, которые являются простыми числами (например, 17, 103, 107) а, затем все остальные элементы массива A. Вывести массив B 10. Найти значение и индекс максимального элемента среди отрицательных элементов массива A

2/ «хорошо»	1. Написать программу транспонирования матрицы 2. Упорядочить строки матрицы по возрастанию значений их наибольших элементов, сформировав вектор-столбец 3. Для заданной матрицы найти сумму положительных и отрицательных элементов по каждой строке и сформировать из этих сумм двумерный массив, который вывести на печать 4. Определить, является ли заданная целая квадратная матрица N-го порядка симметричной относительно главной диагонали 5. Дана матрица размером M*N. Переставив ее строки и столбцы, добиться того, чтобы наибольший элемент оказался в верхнем левом углу 6. Элемент матрицы является седловой точкой, если он является наименьшим в своей строке и одновременно наибольшим в своем столбце. Для заданной матрицы напечатать индексы всех ее седловых точек 7. Определить k - количество "особых" элементов матрицы C, считая элемент "особым", если он больше суммы остальных элементов своего столбца; сформировать массив из особых элементов 8. Дана матрица M на N. Переставив её строки и столбцы, добиться того, чтобы наименьший элемент оказался в верхнем левом углу 9. В матрице размером M*N выбрать и вывести на экран строки, содержащие заданное значение K 10. Задана матрица A размером M*M. Исключить из неё строку и столбец, на пересечении которых находится минимальный элемент
3/ «отлично»	1. Определить функцию, которая сортирует слова в строке в алфавитном порядке (использовать функции strcmp(); strcpy();) 2. Определить функцию, которая оставляет в строке те слова, перед которыми в последовательности находятся только меньшие (по алфавиту) слова, а за ними - только большие (использовать функции strcmp(); strcpy();) 3. Определить функцию, которая удаляет из строки повторные вхождения слов (использовать функции strcmp(); strcpy();) 4. Определить функцию, которая оставляет в строке те слова, которые встречаются в последовательности только один раз (использовать функции strcmp(); strcpy();) 5. Определить функцию, которая считает для каждого слова число его вхождений в строку (использовать функцию strcmp();) 6. Определить функцию, которая оставляет в строке те слова, у которых одинаковые «соседи», т.е. совпадают предыдущее и следующее слова (использовать функции strcmp(); strcpy();) 7. Определить функцию, которая переворачивает каждое слово в строке наоборот, сохраняя последовательность разделителей между словами (использовать функцию strrev();) 8. Определить функцию, которая оставляет в строке те слова, в которых повторно встречается первый символ слова (использовать функции strchr(); strcpy();) 9. Определить функцию, которая исключает из строки слова, содержащие заданную букву (использовать функции strchr(); strcpy();) 10. Определить функцию, которая увеличивает длины всех слов в строке до N символов, дублируя последнюю букву, где N — длина самого длинного слова в строке (использовать функцию strlen();)

В течение семестра дважды проводится промежуточная аттестация по дисциплине: тестирование на 7 и 12 учебных неделях (контрольные недели). Примеры теста №1 и теста №2 приведены в приложениях к ФОС.

Форма экзаменационного билета

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
Факультет АВТФ

Билет № 10
к экзамену по дисциплине «Информатика»

1. Переменные адреса, ссылки. Операции со ссылками. Примеры.
2. Найти значение и индекс максимального элемента среди отрицательных элементов массива A.
3. В матрице размером M*N выбрать и вывести на экран строки, содержащие заданное значение K (уровень сложности 2).

Утверждаю: зав. кафедрой ____ _____ профессор Гриф М.Г.
(подпись)

(дата)

2. Критерии оценки

Ответ на экзаменационный билет считается **неудовлетворительным**, если студент не дает ответа на вопрос и не может решить задачу уровня сложности 1, оценка составляет 24-49 баллов.

Ответ на экзаменационный билет засчитывается на **пороговом** уровне, если студент полностью справился с теоретическим вопросом и решил задачу уровня сложности 1, ответил на дополнительные вопросы, оценка составляет 50-66 баллов.

Ответ на экзаменационный билет засчитывается на **базовом** уровне, если студент полностью справился с теоретическим вопросом, решил задачу уровня сложности 1 и 2, ответил на дополнительные вопросы, оценка составляет 67-86 баллов.

Ответ на экзаменационный билет засчитывается на **продвинутом** уровне, если студент полностью справился с теоретическим вопросом, решил задачу уровня сложности 1 и 3, ответил на дополнительные вопросы, оценка составляет 87-100 баллов.

3. Шкала оценки

В общей оценке по дисциплине экзаменационные баллы учитываются в соответствии с правилами балльно-рейтинговой системы, приведенными в Таблице 2.

Таблица 2

Виды учебной деятельности	Балл максимальный	Балл минимальный
Операторы описания. Условные оператор. Тернарная операция.	3	2
Операторы цикла. Операторы передачи управления.	9	5
Оператор-переключатель. Перечисляемые типы.	3	2
Массивы целых чисел.	5	3
Символьные массивы – строки.	5	3
Указатели.	5	3
Двумерные массивы – матрицы.	3	2
Массивы указателей.	10	6

Массивы структур. Структуры данных в бинарных файлах.	12	8
Текстовые файлы.	5	3
РГЗ	10	5
Экзамен	40	20
Итого баллов	100	62

В таблице 3 представлена шкала перехода от 100 - балльной шкалы к системе оценок ECTS и к традиционной шкале оценок.

Таблица 3

<table>
<tr><th>Диапазон баллов рейтинга</th><th>Оценка ECTS</th><th colspan="2">Традиционная (4-уровневая) шкала оценки</th></tr>
<tr><td rowspan="3">90-100</td><td>A+(97)</td><td rowspan="3">отлично</td><td rowspan="8">зачтено</td></tr>
<tr><td>A(94)</td></tr>
<tr><td>A-(90)</td></tr>
<tr><td rowspan="3">80-89</td><td>B+(87)</td></tr>
<tr><td>B(84)</td><td rowspan="3">хорошо</td></tr>
<tr><td>B-(80)</td></tr>
<tr><td rowspan="3">70-79</td><td>C+(77)</td></tr>
<tr><td>C(74)</td></tr>
<tr><td>C-(70)</td><td>удовлетворительно</td></tr>
<tr><td rowspan="3">60-69</td><td>D+(67)</td><td rowspan="4">удовлетворительно</td><td rowspan="4">зачтено</td></tr>
<tr><td>D(64)</td></tr>
<tr><td>D-(60)</td></tr>
<tr><td>50-59</td><td>E</td></tr>
<tr><td>25-49</td><td>FX</td><td rowspan="2">неудовлетворительно</td><td rowspan="2">не зачтено</td></tr>
<tr><td>0-24</td><td>F</td></tr>
</table>

4. Вопросы к экзамену по дисциплине «Информатика»

1. Предмет и задачи информатики.
2. Информация и информатика. Типы основных операций с данными.
3. Управляющие структуры: ЦИКЛЫ, ЦЕПОЧКА и ВЕТВЛЕНИЕ.
4. Композиции базовых структур. Причины использования базовых структур.
5. Управляющие структуры: ПОДПРОГРАММЫ. Параметры подпрограмм.
6. Определение и вызов ПОДПРОГРАММ. Формальные и фактические параметры. Причины использования.
7. Структура программы на языке Си. Основные лексемы языка. Примеры.
8. Определение, объявление и вызов функции в языке Си. Примеры.
9. Способы передачи параметров в функции в языке Си. Примеры.
10. Типы данных. Основные принципы концепции типов данных.
11. Простейшие типы данных. Стандартные типы данных в языке Си.
12. Составные типы данных: МАССИВЫ. Преимущества массивов.
13. Переменные адреса, указатели. Операции с указателями. Примеры.
14. Переменные адреса, ссылки. Операции со ссылками. Примеры.
15. Массивы указателей. Преимущества, недостатки. Примеры
16. Составные типы данных: СТРУКТУРЫ. Операции со структурами.
17. Массивы структур. Сходства и отличия массива и структуры. Примеры.
18. Функции и массивы. Способы передачи в качестве параметров и возврата значения.
19. Структуры и функции. Способы передачи в качестве параметров и возврата значения.

20. Типы алгоритмов. Примеры.
21. Способы реализации алгоритмов: Итерация и Рекурсия.
22. Свойства рекурсивных алгоритмов.
23. Методы построения алгоритмов: метод «Разделяй и властвуй» и метод Обратного прохода.
24. Методы построения алгоритмов: метод Последовательных приближений и метод Наискорейшего спуска.
25. Методы построения алгоритмов: метод Поиска с возвратом и метод Выделения подцелей.

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Новосибирский государственный технический университет»
Кафедра автоматизированных систем управления
Кафедра автоматики
Кафедра вычислительной техники

Паспорт
расчетно-графического задания (работы)

по дисциплине «Информатика», 1 семестр

1. Методика оценки

В рамках расчетно-графического задания по дисциплине студент:

- выбирает структуры данных;
- проектирует набор функций;
- разрабатывает пользовательский интерфейс.

При выполнении расчетно-графического задания студент:

- *формулирует* постановку задачи*;*
- *определяет:* схему иерархии программных модулей, функциональное описание каждого модуля, входные/выходные данные;
- *выбирает* структуры данных, форматы данных;
- *строит* блок-схемы обобщенных алгоритмов;
- *оценивает* степень сцепления и связности модулей;
- кодирует, отлаживает и тестирует программу;
- *анализирует* результаты проектирования, кодирования, отладки и тестирования;
- *составляет* расчетно-пояснительную записку объемом 30-40 страниц.

Деятельность обучающегося и критерии оценки ее качества на основных этапах выполнения РГЗ

Таблица 1

Этап	Деятельность обучающегося	Макс. балл	Критерии качества
1	Выбор задания. Описание постановки задачи.	5	своевременность выбора, полнота системного описания
2	Определение и утверждение схемы иерархии программных модулей	5	своевременность – 2, правильность - 3
3	Функциональное описание каждого модуля, определение входной и выходной информации	15	своевременность – 5, правильность, универсальность –10
4	Описание структур данных. Построение блок-схем алгоритмов. Оценка сцепления и связности модулей.	20	своевременность – 5, универсальность – 15
5	Разработка интерфейса, программная реализация	15	своевременность – 5, качество интерфейса – 10
6	Отладка, тестирование программы. Анализ процесса проектирования, кодирования, тестирования	15	• Качество тестирования – 5; • Своевременность–5; • Универсальность- 5

	программы.		
7	Оформление пояснительной записки	15	логичность, связность, полнота описания работы
8	Защита РГЗ	10	качество презентации – 5, качество доклада - 5
Итого		100	

В зависимости от количества набранных баллов за выполнение соответствующих этапов курсовая работа оценивается следующим образом: 85-100 баллов – **отлично** (10 баллов в общей оценке по дисциплине), 70-84 баллов – **хорошо** (8 баллов в общей оценке по дисциплине), 45-69 – **удовлетворительно** (5 баллов в общей оценке по дисциплине), < 45 баллов или отсутствие работающего программного продукта – **неудовлетворительно** (2 балла в общей оценке по дисциплине).

2. **Критерии оценки**

• Работа считается **не выполненной,** если выполнены не все части РГЗ, отсутствует этапы выполнения РГЗ со 2 по 8, оценка составляет 2 балла.

• Работа считается выполненной **на пороговом** уровне, если части РГЗ выполнены не в полном объеме: отсутствуют 4 и 6 этапы выполнения РГЗ, оценка составляет 5 баллов.

• Работа считается выполненной **на базовом** уровне, если выполнены все этапы, но 4 и 5 этапы выполнены неэффективно, оценка составляет 8 баллов.

• Работа считается выполненной **на продвинутом** уровне, если все этапы выполнены в полном объеме и защита РГЗ прошла успешно, оценка составляет 10 баллов.

3. Шкала оценки

В общей оценке по дисциплине баллы за РГЗ учитываются в соответствии с правилами балльно-рейтинговой системы, приведенными в таблице 2.

Таблица 2

№	Вид учебной деятельности	Мак. балл
1	Лабораторные работы №1-8.	50
3	РГЗ	10
4	Экзамен	40
	Итого	100

Таблица соответствия баллов, традиционной оценки и буквенной оценки ECTS приведена в рабочей программе дисциплины.

4. **Примерный перечень тем РГЗ**

1. Задано время в часах и минутах. Определить, через сколько минут часовая и минутная стрелки будут образовывать угол в 90 градусов.
2. Задано время в часах и минутах. Определить, через сколько минут часовая и минутная стрелки будут образовывать угол в 45 градусов.
3. Написать программу перевода чисел из арабской записи в римскую запись: *I – 1, V – 5, X – 10, L – 50, C – 100, D – 500, M – 1000.*
4. Написать программу перевода числа из римской записи в арабскую.
5. Напечатайте k-тую степень основания n в десятичной системе счисления, например, k = 50, n = 2, результат 2^{50} = 1125899906842624 (значение хранится в символьном массиве - строке).

6. Напечатайте k-тую степень основания n в восьмеричной системе счисления, например, k = 18, n = 13, результат 13^{18} = 46450626533031772247 1 (значение хранится в символьном массиве - строке).
7. Напечатайте k-тую степень основания n в шестнадцатеричной системе счисления, например, k = 23, n = 12, результат 23^{12} = 2EE56725F06E5C71 (значение хранится в символьном массиве - строке).
8. Создать таблицы сложения и умножения для 12,14,16 - надцатеричной системы счисления, продемонстрировать работу с таблицами. При умножении выводить значения промежуточных результатов (слагаемых).
9. Создать таблицы сложения и умножения для n - ричной системы счисления (n = 3...9), продемонстрировать работу с таблицами. При умножении выводить значения промежуточных результатов (слагаемых).
10. Напечатать на экране сложение и умножение «столбиком» двух чисел, заданных в шестнадцатеричной системе счисления (без перевода в десятичную систему счисления).
11. Напечатать на экране сложение и умножение «столбиком» двух чисел, заданных в восьмеричной системе счисления (без перевода в десятичную систему счисления).
12. Сложить два числа, записанные в римской системе счисления (без перевода в десятичную систему счисления).
13. Найти разность двух чисел, записанных в римской системе счисления (без перевода в десятичную систему счисления).
14. Напечатать на экране сложение и умножение «столбиком» двух длинных целых чисел (значение числа хранится в символьном массиве - строке). При умножении выводить значения промежуточных результатов (слагаемых).
15. Напишите программу, которая позволяет осуществить сложение, вычитание, умножение и деление для длинных целых чисел (значение числа хранится в символьном массиве - строке).
16. Напишите программу, которая позволяет осуществить сложение, вычитание, умножение и деление для длинных целых чисел (значение ЧИСЛА хранится в массиве целых чисел – каждая цифра ЧИСЛА хранится в отдельном элементе массива целых чисел).
17. Напишите программу, которая позволяет осуществить сложение, вычитание, умножение и деление для длинных целых чисел (значение ЧИСЛА хранится в массиве целых чисел – каждые девять разрядов ЧИСЛА хранятся в отдельном элементе массива целых чисел).
18. Определить, имеют ли общие точки *две плоские фигуры* – треугольник с заданными координатами его вершин и круг заданного радиуса с центром в начале координат.
19. Определить, сколько общих точек имеют *периметр* треугольника с заданными координатами его вершин и окружность заданного радиуса.
20. Автоморфными называются числа, которые содержатся в последних разрядах их квадрата, например, десятичные числа: 5^2 = 25, 25^2 = 625. Автоморфные числа существуют в системе счисления, основание n которой не должно быть простым числом или его степенью (n = 6,9,10,12,14…). Составьте алгоритм нахождения k автоморфных чисел в заданной системе счисления.

1 семестр ПРЕД_ТЕСТ №1

Дополните определения (в заданиях 1-20)

1. В повседневной жизни мы используем _____ систему счисления. Это _____ система счисления.
2. Данные и команды внутри компьютера хранятся в _____ коде.
3. Восьмеричная и _____ запись есть компактная форма внутреннего представления чисел.
4. Наименьшим адресуемым участком памяти является _____.
5. Целые числа хранятся в виде знака и абсолютного значения или _____ числа до 1.
6. Самый старший бит обычно является знаковым, остальные называются _____.
7. Действительные числа хранятся в виде двух целых чисел: _____ и _____.
8. Компьютерная переменная – это _____ местоположение в памяти для хранения данных _____ типа.
9. Данные могут быть информацией, вводимой пользователем, _____ результатом вычислений или _____ результатом некоторой операции.
10. Диапазон значений величин - _____ величин от минимального до максимального, который может быть представлен в заданном количестве _____.
11. Лексемы - _____ словаря языка.
12. Язык С распознаёт лексемы шести типов: ключевые слова, _____, константы, строковый литерал, _____, знаки препинания.
13. Ключевыми называются слова, _____ для специальных целей, которые не должны использоваться в качестве _____.
14. Для определения основных типов данных используются следующие ключевые слова: _____ (символьный), short (короткий целый), int (целый), _____ (длинный целый), float (вещественный), _____ (вещественный с удвоенной точностью) …
15. Инициализировать переменную значит присвоить её специфическое _____ _____ сразу после того, как только переменная создана.
16. Ключевые слова signed и unsigned указывают как _____ старший бит переменной.
17. Переменная любого типа может быть объявлена как немодифицируемая с помощью ключевого слова _____, она называется _____.
18. Операторы определяют действия и _____ выполнения этих действий в программе.
19. Операторы можно разделить на 5 групп: описания, _____, передачи _____, вызова _____, пустой оператор.
20. Способы улучшения читаемости программ:
    - _____ имен переменных и функций;
    - использование _____;
    - использование _____ _____ для отделения одной части функции, соответствующей некоторому семантическому блоку, от другой;
    - запись каждого _____ на отдельной строке.

Выберите *все* верные утверждения (назовите буквы)

21. Байт – это:

A. единица измерения информации;
B. минимальный размер файла;
C. составляющая машинного слова, полуслова;
D. 8 бит;
E. участок памяти, не имеющий границ.

Ответ _____.

Выберите *верный* ответ (в заданиях 22,23)(назовите буквы)

22. Килобайт – это

A. 1020 байтов;
B. 1000 байтов;
C. 1021 байт;
D. 1024 байта;
E. 1/10 Мегабайта.

Ответ ___.

23. При следующей записи

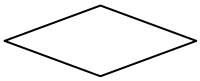
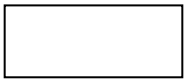
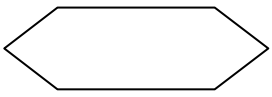
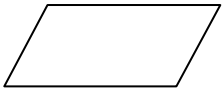
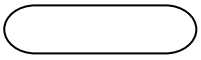
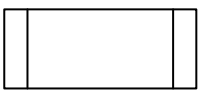
```
if (5 > 3) оператор_1;
     else
оператор_2;
оператор_3;
```

выполнятся операторы:

A. Только оператор 1;
B. Операторы 1 – 3;
C. Операторы 2, 3;
D. Только оператор 2;
E. Операторы 1, 2;
F. Только оператор 3;
G. Операторы 1, 3.

Ответ ___.

24.**Установите соответствие**

Обозначение блоков	Наименование блоков
1. 	**A.** пуск – останов **B.** *процесс* **C.** ввод-вывод **D.** *решение* **E.** модификация **F.** *ресурс* **G.** предопределенный процесс **H.** *начало-конец* **I.** цикл
2. 	
3. 	
4. 	
5. 	
6. 	

Ответ 1___ 2___ 3___ 4___ 5___ 6___.

Установите соответствие (в заданиях 25,26)

25.

Назначение	Компоненты
1) Минимальный набор устройств, необходимых для работы компьютера. **2)** Дополнительные устройства, подключаемые к компьютеру.	**A.** клавиатура **B.** мышь **C.** сканер **D.** модем **E.** системный блок **F.** принтер **G.** монитор **H.** джойстик

Ответ 1) _____ 2) _____.

26.

Операторы	Примеры
1. цикл с постусловием 2. условный 3. цикл с параметром 4. возврат из функции 5. перехода к следующей итерации цикла 6. переключатель 7. выход из цикла 8. цикл с предусловием	**A.** if (a > b) min = b; else… **B.** switch (ch) { case '0':…} **C.** while (k-- > 0) f = f*k; **D.** do {f=f*k;k--;} while (k>0); **E.** for (f = 1; k > 0; f = f*k, k --); **F.** return; **G.** break; **H.** continue;

Ответ 1___ 2___ 3___ 4___ 5___ 6___ 7___ 8___.

Выберите *верный* ответ (в заданиях 27,28)(назовите букву)

27.В операторной записи

```
if (n = = 5) оператор_1;
    else
оператор_2;
```

(n = = 5) *является:*

A. логическим выражением, которое принимает значение "истинно";
B. логическим выражением, которое присваивает переменной n значение 5;
C. логическим выражением, которое принимает значение "истинно" либо "ложно";
D. логическим выражением, которое принимает значение "истинно", если n не равно 5, и "ложно", если n равно 5.

Ответ ___.

28.Использование оператора break **не**допустимо в следующей записи:

A. for (вып1; вып2; вып3)
{
оператор1; оператор2;
if (условие) break;
оператор3;
}
	B. while (выражение)
	оператор1;
	оператор2;
	if (условие) break;
	оператор 3;
		C. do
		{
		if (условие) break;
		else оператор;
		} while (выражение);

Ответ ___.

Выберите *верный* ответ (в заданиях 29-31)(назовите букву)

29. Для строки «В байте 8 бит.» выделяется объем памяти, равный:
    A. 16 байтов;
    B. 15 байтов;
    C. 128 бит;
    D. 17 байтов.

Ответ ___.

30. Истинным является выражение –
    A. ((5>3)&&(4= =7))
    B. ((4!=7)&&(5<=3))
    C. ((6= =7) || (5<=5))
    D. ((5>=7) || (3!=3))

Ответ ___.

31. Массив – это:
    A. множество переменных одного типа;
    B. область памяти с одним именем, содержащая несколько значений одного типа;
    C. несколько переменных разного типа;
    D. область памяти с одним именем, содержащая несколько значений разного типа;
    E. последовательность чисел, содержащихся в последовательных ячейках памяти.

Ответ ___.

Выберите *неверный* ответ (в заданиях 32,33)(назовите букву)

32. Бесконечные циклы
    A. while (5 != 5) puts ("End line");
    B. for (i =1; i >=0; i + i) puts ("End line");
    C. do{ puts ("End line");} while(5= =5);

Ответ ___.

33. Способы определения строки
    A. char str1[] = "Строка №1";
    B. char *str_2 [2] = {"Строка 1", "Строка 2"};
    C. char *Str3 = "строка 3";
    D. #define String "строка 1"
    E. char string[] = { 'S', 't', 'r', 'i', 'n', 'g', '\0' };

Ответ ___.

34. **Установите соответствие**

Тексты программ	Результаты работы

1. #include <stdio.h>
#include <conio.h>
void main()
{
for (int i = 0; i <= 3; i++);
printf ("%d ", i);
getch();
}
	A. 0 1 2 3
	B. 3
	C. 2
	D. 4
	E. 0 1 2

2. #include <stdio.h>
void main()
{
int a[5] = {3,2,0,1,4},
b[5] = {3,2,0,1,4};
for (int i = 0; i < 5; i++)
printf ("%d ", a [b[i]]);
}
	A. 1 0 2 3 4
	B. 3 2 0 1 4
	C. 1 0 3 2 4
	D. 0 2 4 3 1
	E. 1 0 2 4 3

Ответ 1___ 2 ___.

Составитель - Смирнова Анна *группа АС-713*

36. Что такое switch?

37. Что такое рекурсия?

38. Как поступают, если в программе необходимо работать не с одним объектом, а с последовательностью объектов?

39. Как можно подсчитать количество положительных элементов массива?

40. Каким образом можно осуществить подсчет суммы положительных элементов массива?

41. Как найти номер максимального элемента в массиве?

42. Что называется динамической памятью, динамическим массивом?

43. Каким образом можно написать программу, вычисляющую определитель квадратной матрицы второго порядка? Каким образом вообще можно работать с матрицами?

44. Как можно поменять местами элементы массива, чтобы оказалось, что последующий элемент больше предыдущего?

45. Как найти номер максимального элемента в массиве?

46. В каком порядке выполняются операции в выражении? Можно ли изменить этот порядок?

47. Какие операции относятся к логическим? Для чего они нужны?

48. Для чего нужно разделение программ по функциям?

49. Для чего нужны сортировки, какими они бывают? Что такое ключ?

50. Что такое сортировка методом “пузырька”?

Составитель - Смирнова Анна *группа АС-713*

1. Если С – это язык, то есть ли у него свои слова и словосочетания?

В этом языке, как и в любом другом, есть свои слова, которые называются лексемами, словосочетания, называемые выражениями и предложения – операторы.

Лексемы образуются из символов, выражения – из лексем и символов, операторы – из символов, выражений и лексем.

Алфавит этого языка (основные неделимые знаки, с помощью которых пишутся все тексты на языке) состоит из:

- прописных, строчных букв латинского языка и знака подчеркивания;
- арабских цифр от 0 до 9;
- специальных знаков (например, [ ], ( ), %, # и т.д.);
- пробельных символов.

Из этих “букв” составляются “слова” – лексемы, которые разделяются на :

- идентификаторы (имена объектов), Например: SYMBOL, imia и т.д.;
- ключевые слова (идентификаторы, имеющие особое значение для компилятора). Например, for, while, else и т.д.;
- знаки операций (один или более символов, определяющих действие над операндами). Например, +, *, >= и т.д.;
- константы (неизменяемые величины);
- разделители (( ), ‘.’, ‘;’, ‘ ‘).

Выражения состоят из операндов, знаков операций и скобок. Примеры:

(a+7)/6
x&&y||!z
(a*sin(x)-1.05e4)/((2*k+2)*(2*k+3))

2. Почему все программы начинаются с одной и той же строчки?

Все программы начинаются со строки: #include ‹stdio.h›.

#include называется директивой препроцессора. Она вставляет содержимое указанного файла в ту точку исходного файла, где она записана.

stdio.h является заголовочным файлом для функций ввода и вывода в языке С, к которым относятся, например, printf (выводит строку параметров), scanf (вводит строку параметров) т.д. А так как действующая программа вряд ли может существовать без функции ввода и вывода (как иначе будет осуществляться связь между пользователем и программой?) то, следовательно, строка #include ‹stdio.h› необходима.

#include – это простейшее средство обеспечения согласованности объявлений в различных файлах. Директива обеспечивает включение в них информации о пользовательском интерфейсе из заголовочных файлов (кроме ‹stdio.h› это могут быть, например, ‹conio.h›, ‹math.h› (заголовочный файл, содержащий прототипы математических функций).

3. Роль знака # ? Когда вводится #? Что такое директивы, препроцессор, компилятор? Как они связаны?

Все директивы препроцессора начинаются со знака #.

Пример: #include <stdio.h>.

Препроцессор – это первая фаза компилятора.

Компилятор – это обслуживающая программа, которая осуществляет перевод программы с языка программирования на машинный язык (язык, понятный процессору). Препроцессор выполняет директивы (инструкции).

Директивы должны начинаться с символа #, перед которым в строке могут находиться только пробельные символы.

После выполнения препроцессором директив, текст программы поступает на вход компилятора, который выделяет лексемы (минимальные единицы языка программирования), а затем распознает выражения и операторы, построенные из этих лексем. При этом компилятор определяет синтаксические ошибки и, если их нет, строит объектный модуль. Далее компоновщик формирует исполняемый модуль программы, подключая к объектному модулю другие объектные модули. Если программа состоит из нескольких исходных файлов, то они компилируются по отдельности и объединяются на этапе компоновки.

4. Как в программе можно проверить, что вводимое число является натуральным?

Натуральные числа – числа, с помощью которых можно осуществить счет предметов. Следовательно, в записи программы надо отразить, что введенное число является целым, и оно больше нуля.

Сделать это можно несколькими способами:

1. Используя беззнаковый тип данных unsigned int.
2. Пользуясь следующей записью:

```
const int min = 1, max = 32767;
unsigned int n;
do
{
printf("Введи число в диапазоне от %d до %d", min, max);
scanf("%u", &n);
}
while ((n ‹ min)  || (n › max));
…
```

В этой записи говорится о том, что программа выполняется только тогда, когда введенное число n больше нуля и ≤ 32767 (32767 – это верхняя граница значений целого типа int со знаком).

5. Имеется задача, в которой требуется определить, расположены ли цифры в записи натурального числа симметричным образом. Как можно определить количество разрядов в числе?

Количество разрядов числа можно определить, пользуясь следующей функцией

```
int Razriad_chisla (long n)
{
   int k=1;
   while ((n/(long) pow (10, k)) ›0) k++;
   return k;
 }
```

В этой функции k – количество разрядов, n – само число. Цикл подсчета разрядов начинается с 1 (число разрядов не может быть меньше 1) и продолжается

до тех пор, пока выполняется условие, что частное числа n и 10 в степени k больше нуля. Пока это условие соблюдается, k последовательно увеличивается на единицу (операция инкремент). Когда это условие перестает выполняться, то возвращается полученное значение k.

6.Какие моменты важно учесть при объявлении переменных?

Каждая переменная должна быть объявлена, иначе компилятор выдаст ошибку.

Переменные объявляются в любом месте программы, но обязательно до их использования. Обычно объявляются в начале функции, сразу за заголовком.

Пример:

```
void Mini_Max (int Array [], int size, int &max)
{
int count = 0, sum = 0;
...
}
```

Идентификатор – это имя программного объекта. Он может состоять из латинских букв, цифр и знаков подчеркивания. Но первым символом обязательно должна быть буква. Пробелы внутри имен не допускаются, вместо них используется знак подчеркивания.

Например: int a b;- неправильно. int a_b;-правильно.

Компилятор различает прописные и строчные буквы, поэтому, если вместо SIMBOL, объявленного вначале, напечатать Simbol или symbol, то компилятор выдаст ошибку, так как не обнаружит эти имена. Также идентификатор не должен совпадать с ключевыми словами. Не рекомендуется начинать имя со знака подчеркивания.

Решение любых задач следует начинать с описания того, какие переменные нам понадобятся, но иногда приходится их добавлять по ходу решения задачи.

Пример: объявить переменные, необходимые для вычисления стоимости покупки, состоящей из нескольких тетрадей и одной книги

```
float Cena_T, Cena_Kn; // тип float, т.к. цены тетрадей и книги могут быть дробным числом
int kol_T;  // int, т.к. количество тетрадей может быть только целым числом
```

7. Когда употребляется int main(), а когда void main()?

Функция main() – это функция, благодаря которой начинает осуществляться действие программы. Без нее программа не запустится.

Если функция main() получает параметры и возвращает результат, то она объявляется int main ().

Примером такой программы может послужить программа, запрашивающая у пользователя число и выводящая его на экран.

```
#include ‹stdio.h›
int main ()
{
int i;
printf ( "Введите число: ");
scanf ("%d", &i);
```

```
printf ( "Вы ввели число %d, спасибо!", i);
return 0;
}
```

В том случае, если функция main не получает параметры и не возвращает результат, то она объявляется: void main ().

Программа с использованием void main() может выглядеть следующим образом:

```
#include ‹stdio.h›
# include ‹conio.h›
void main ()
{
printf ("Привет!");
getch ();
}
```

8. Какие типы данных наиболее часто используются в программах? Чем они отличаются?

Пожалуй, наиболее часто используются типы int и float.

Тип int – целочисленный тип данных. Он может быть использован, например, в случае, если переменной присваивается количество чего либо, или в любом другом случае. Когда в задаче указано, что число должно быть целым.

Например, если требуется объявить переменные, необходимые для подсчета общего количества покупок, если приобретено несколько книг и несколько открыток. Запись:

```
int a = 0, b = 0; // количество может быть только целой величиной.
```

Тип float – тип с плавающей точкой. Он используется, когда данное число или предполагаемый результат могут быть дробными.

Например, если требуется объявить переменные, необходимые для вычисления площади прямоугольника, то запись будет выглядеть так:

```
float a, b, s;// ширина, длина и площадь могут быть дробными числами.
```

Эти типы различаются диапазоном значений:

для типа float он: 3.4 e-38…3.4e+38,

а для типа int зависит от спецификаторов.

Так, если его спецификатор – long, то он занимает диапазон от (-2147483647) до +2147483647, а если short – от (-32767) до +32767.

Также, они различаются размером. Тип float занимает 4 байта. Размер типа int не определяется стандартом, а зависит от компьютера и компилятора (может занимать 2 или 4 байта).

Отличаются эти типы и внутренним представлением в компьютере. Величина типа int представлена как целое число в двоичном коде, а величина типа float будет состоять из двух частей: мантиссы и порядка.

Величина типа int может быть переведена в тип float без потери точности, при переводе же из типа float в тип int точность может быть потеряна (будет потеряна дробная часть).

9. Что такое функция printf()? Для чего она нужна?

Функция printf() – функция вывода на экран монитора сообщений и значений переменных, а так как без этого невозможна никакая программа, то функция эта очень важна и используется во всех программах.

Первым параметром этой функции является строка вывода, определяющая выводимый текст и формат вывода значений переменных, который задается при помощи спецификаторов преобразования последовательности символов, начинающихся с символа %.

Наиболее часто используемые спецификаторы:

% d - для вывода целых со знаком;

% u - для вывода беззнаковых целых;

% f - для вывода дробных, в виде числа с плавающей точкой.

Существуют определенные сочетания символов, означающие выполнение определенных действий: “ \n” – новая строка, "\t" – табуляция, и т.д.

Пример: Записать фрагмент программы, подсчитывающей количество покупок, если куплено несколько игрушек и несколько книжек (выводящий на экран параметры, вводимые пользователем).

```
printf (“a = % d  b = % d,” a, b); // где a – количество игрушек,
                                   b – количество книжек.
```

Записать фрагмент программы, выводящей на экран параметры, вводимые пользователем, необходимые для вычисления площади прямоугольника:

```
printf (“a  = % f  b = % f,” a, b) ;     // a  – ширина, b – длина
printf (“a  = % f  b = %f  \n”, a, b); // после ввода этих параметров курсор переводится на следующую строку.
```

10. Для чего предназначена функция scanf()? Что означают символы в кавычках и после знака &?

Функция scanf() предназначена для ввода исходных данных с клавиатуры.

Первым параметром функции scanf() является управляющая строка.

Управляющая строка – это список спецификаторов, заключенных в кавычки:

%d - для ввода целых чисел,

%f - для ввода дробных чисел,

%u – для ввода целых беззнаковых чисел,

%c – для ввода символа,

%s – для ввода строки.

Остальные параметры – адреса переменных, значения которых должны быть введены. Типичной ошибкой является использование имени переменной, в то время как должен быть использован адрес (кстати, компилятор эту ошибку не обнаруживает).

Пример:

Написать инструкцию, которая обеспечивает ввод значений дробных переменных a и b:

```
scanf (“%f %f ”, &a, &b);
```

Т.к. переменные по условию должны быть дробного типа, то используется спецификатор %f, запись &a, &b означает, что используются адреса переменных a и b, а не сами эти переменные.

Объявить необходимые переменные и написать инструкции ввода исходных данных для программы вычисления стоимости покупки, состоящей из нескольких тетрадей и карандашей.

```
float c_tetr, c_kar; // цена тетради и карандаша
int n_tetr, n_kar;   //  количество тетрадей и карандашей
printf (“Введите цену и количество \n “);
printf (“Тетради:”);
scanf (“%f %d”, &ctetr, &ntetr);
printf (“Карандаши:”);
scanf (“%f %d”, &ckar, &nkar);
```

Т.к. цена может быть дробным числом, то используется %f, т.к. количество может быть только целым числом, то используется %d.

Заголовочный файл для функции scanf() - stdio.h, т.е. в программе, где используется эта функция, обязательно должна в начале быть строка #include<stdio.h>.

11. Что означает запись max = (b> a)? b:a; в программе?

```
#include ‹stdio.h›
int main ( )
{
int a = 11, b = 4, max;
max = (b > a) ? b : a;
 printf ("max = % d", max);
return 0;
}
```

В этой программе присутствует тернарная операция (? :). Называется она так потому, что в ней присутствует три операнда и ее общий вид: операнд_1? операнд_2: операнд_3.

Первый операнд рассматривается с точки зрения истинности (верно - true, неверно - false). Если результат вычисления - true, то результатом всей операции будет значение второго операнда, а если false – то третьего.

В данной задаче на экран будет выведено "max = 11", т.к. 11 больше 4.

По-другому эту программу можно записать так:

```
#include ‹stdio.h›
 int main ( )
{
int a = 11, b = 4, max;
if (b > a)   max = b;
        else max = a;
printf (“max = %d”, max);
return 0;
}
```

12. Что такое указатель? Для чего нужны указатели?

Передавать данные можно двумя способами:

1. создавать в каждой точке программы копию тех данных, которые необходимо обрабатывать:

2. передавать информацию о том, где в памяти расположены данные.

Естественно, второй способ удобнее, а запись получается компактнее.

Указателем называется переменная, содержащая информацию о расположении в памяти другой переменной, т.е. указатели предназначены для хранения адресов областей памяти. Следовательно, основная операция для указателя – косвенное обращение к той переменной, адрес которой он содержит.

Имеется специальная операция "*", которая означает переход от переменной-указателя к той переменной, на которую она ссылается. Для доступа к величине, адрес которой хранится в указателе, применяется операция разыменования.

Пример:

```
char a, *b = new char; /* выделение памяти под указатель и динамическую переменную типа char*/
*b = 'A';
a  = *b ; //  присвоение значения обеим переменным.
```

Последовательность действия с указателями следующая:

1. Определение "указуемых" переменных и переменной указателя:

```
int  a, x = 50;
int * p;
```

2. Необходимо связать указатель с "указуемой" переменной, указатель должен быть назначен на переменную, на которую он ссылается.

```
p = &a; // указатель содержит адрес переменной a.
```

3. Выполнение требуемых действий тот же участок программы без использования указателя

```
*p = 100; / /эквивалентно a = 100;
x = x + *p; //эквивалентно x = x + a;
(*p) ++; // эквивалентно a ++;
```

Различают три вида указателей: указатель на объект, указатель на функцию и указатель типа void.

13. Можно ли выполнять какие – либо операции с указателями? Если да, то есть ли ограничения?

Да, с указателями можно выполнять следующие операции: сложение с константой, вычитание, инкремент и декремент.

Инкремент перемещает указатель к следующему элементу массива, декремент – к предыдущему.

Сложение указателя с константой означает, что его значение изменяется на величину этой константы, умноженную на размер объекта данного типа.

Пример:

```
short *a = new short [5];
a ++; // значение переменной a увеличивается на 2, т.к. спецификатор short занимает 2 байта.
```

Разность двух указателей понимается как расстояние между ними.

Суммирование двух указателей не допускается, т.к. не имеет смысла.

Существуют и ограничения на выполнение операций с указателями: операции применимы только к указателям одного типа и имеют смысл в основном при работе с последовательно размещенными в памяти структурами данных, например, с массивами.

14. Чем отличается ссылка от указателя?

Ссылка – это синоним имени, указанного при инициализации ссылки, в то время, как указатель – это переменная, хранящая адрес другой переменной.

Пример ссылки:

```
int rol;
int &elf  =  rol;  // ссылка elf – другое имя для rol.
```

Нельзя осуществить ссылку на ссылку, в то время как можно создать указатель на указатель (**p).

Ссылка, в отличие от указателя, не занимает дополнительного места в памяти, т.к. является просто другим именем величины. Для ссылки не надо выделять память, и операция над ссылкой приводит к изменению величины, на которую она ссылается.

15. Что такое адрес, адресация?

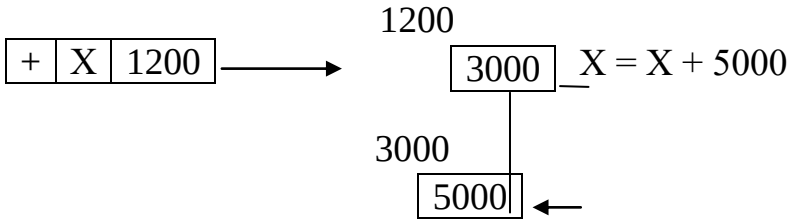
Внутренняя память (так называемая, оперативная память) компьютера представляет собой упорядоченную последовательность байтов или ячеек памяти, проще говоря – массив. Номер ячейки памяти, через который она доступна для команд компьютера и во всех других случаях, называется адресом.

Если в команде непосредственно содержится адрес памяти, то такой доступ к этой ячейке называется прямой адресацией. Но возможен и другой случай: когда доступ к ячейке осуществляется через другую ячейку (случай с указателями). Это называется косвенной адресацией.

Пример:

прямая адресация

косвенная адресация



16. Для чего нужна операция new? Как с ней оперировать?

Операция new - одна из операций выделения памяти и присвоения ее адреса указателю, т.е. это один из способов инициализации указателей.

Примеры:

```
int *n = new int (10); //1
int *a = new int [10]; //2
```

В первом случае с помощью new выделяется память под величину типа int и адрес начала этого участка записывается в переменную n. Затем производится инициализация выделения динамической памяти значением 10.

Во втором случае операция new выполняет выделение памяти под 10 величин типа int и записывает адрес начала этого участка в переменную a, которую можно назвать именем массива. Через это имя можно обращаться к любому элементу массива.

Память, выделенная с помощью операции new, должна быть освобождена. Выполняется это с помощью delete.

Пример:

```
delete n;
delete [ ] a;
```

Существуют так же функция выделения динамической памяти, например, malloc().

Пример:

```
int *m = (int*) malloc (sizeof  (int)); //3
```

В примере 3 выполняется то же, что и в примере 1 (без инициализации 10).

Освобождение памяти, выделение с помощью malloc() происходит с использованием free.

Пример: free (m);

Использовать new предпочтительнее, особенно при работе с объектами классов.

17. Существуют ли другие способы инициализации указателя (кроме выделения участка динамической памяти и присвоения ее адреса указателю)? Какие?

Да, существуют следующие способы:

1. Присвоение указателю адреса существующего объекта:

а) с помощью операции получения адреса:

```
int a = 5;
int *p = &a; // в указатель записывается адрес a
```

б) с помощью значения другого указателя:

```
int *r = p;
```

в) с помощью имени массива или функции, которые трактуются как адрес

```
int a[10];
int* k = a; // присвоения адреса начала массива

void f (int l) {/*...*/}
void (*pf) (int);  // указатель на функцию
pf = f;              // присвоение адреса функции
```

2. Присвоение указателю адреса области памяти в явном виде

```
char *vp = (char*) 0xB80000;
```

0xB80000 – шестнадцатеричная константа, a (char*) - операция явного приведения типа, т.е. константа преобразуется к типу “указатель на char”.

3. Присвоение пустого значения

```
int * SUXX = NULL;
int * ruler = 0;
```

Но рекомендуется использовать просто 0, а не NULL, т.к. это значение типа int будет правильно преобразовано обычными способами в соответствии с контекстом.

Т.к. гарантируется, что объектов с нулевым адресом нет, то пустой указатель можно использовать для проверки, ссылается указатель на конкретный объект или нет.

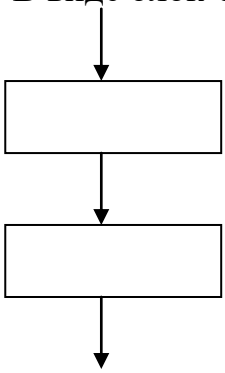
18. Что такое базовые конструкции? Для чего они нужны? Какие бывают базовые конструкции? Что такое следование?

Любую программу для решения задачи можно составить только из трех структур, называемых "следование", "ветвление" и "цикл". Эти структуры и называются базовыми конструкциями.

Целью использования базовых конструкций является получение программы "простой структуры". Такую программу проще читать, отлаживать и вносить в нее изменения. Особенностью базовых конструкций является то, что любая из них имеет только один вход и один выход. Поэтому конструкции могут вкладываться друг в друга произвольным образом. И программа может состоять из различного числа базовых конструкций и может иметь какой угодно по сложности вид.

Следование – это конструкция, которая представляет собой последовательность выполнения двух или более операторов (простых или составных).

В виде блок-схемы эту конструкцию можно записать так:



В этой программе все действия выполняются одно за другим, последовательно, как бы по прямой с линейной структурой, в ее написании прослеживается конструкция следования.

Пример программы с линейной структурой (следование):

Написать программу, вычисляющую скорость бегуна на дистанции.

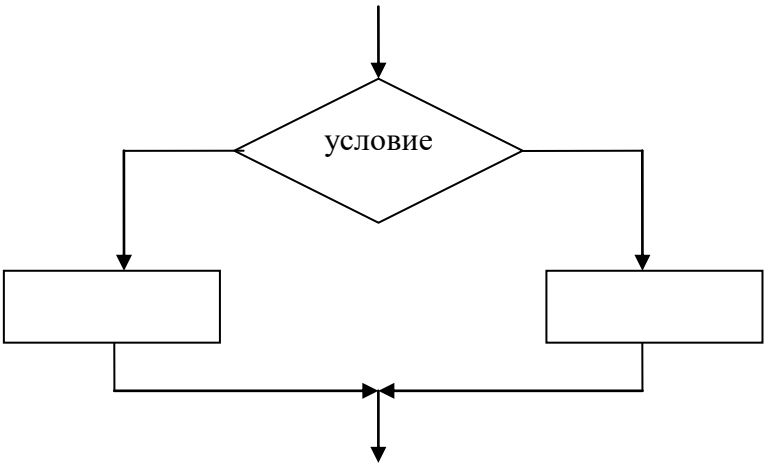
```
#include<stdio.h>
void main ( )
{
 float s, t, v, t_s; // s – дистанция, t – время, v - скорость,
                     //t_s – время в секундах
 int min, sek;      // min – минут, sek – секунд
 printf ("Вычисление скорости бега  \n");
 printf ("Введите длину дистанции и время \n");
 scanf ("%f  %f", &s, &t);
        min = t;
        sek = (t – min)*100;
        ts = min*60 + sek;
        v = (s/1000) / (t_s/3600);
```

```
 printf (“дистанция - % f”, s);
 printf (“время - % d минут % d секунд = %Fсекунд  \n”, min, sek, t_s);
 printf (“скорость бега  %f км/час \ n”,v);
}
```

19. Что собой представляет базовая конструкция ветвление?

Ветвление – конструкция, которая задает выполнение либо одного, либо другого оператора в зависимости от выполнения какого-либо условия.

Фрагмент блок-схемы "ветвление"



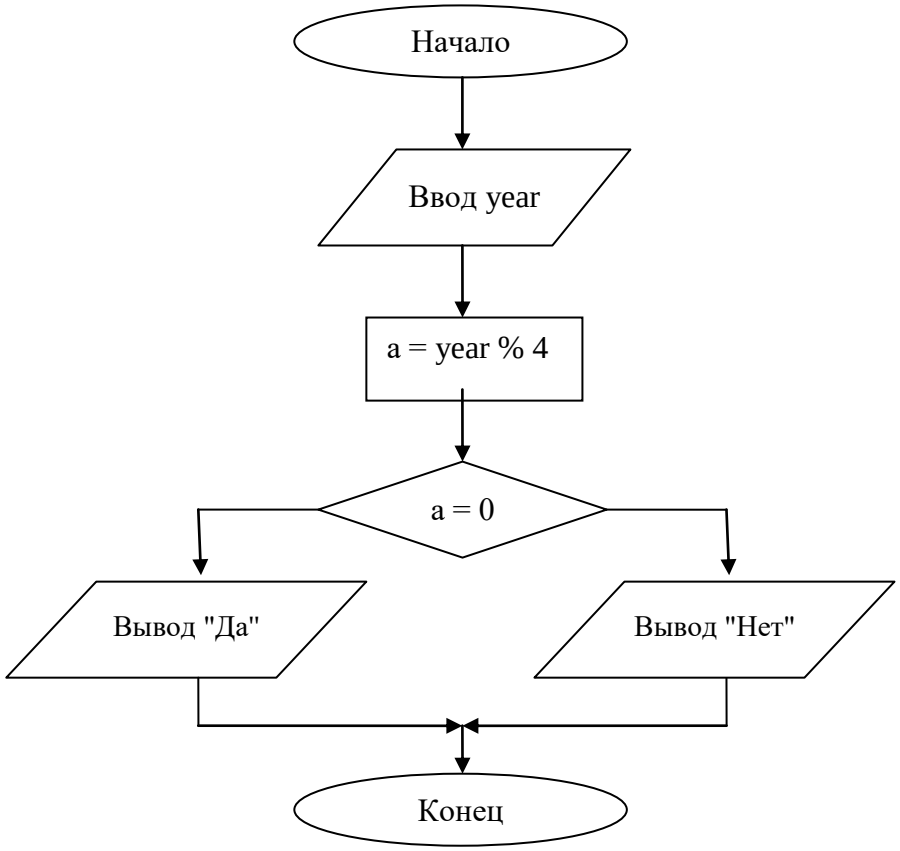
Пример:

Написать программу, которая проверяет, является ли год високосным?

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
int year,a; // a – остаток от деления на 4.
printf (“Введите год\n”);
scanf (“% d”, &year);
a = year % 4;
if (a = = 0)        printf (“% d – високосный год \n”);
else                printf (“% d – не високосный год \n”);
getch ( );
}
```

Т.е., если остаток после деления года на 4 равен нулю, то выполняется одно действие – выводится сообщение, что год високосный, если условие не выполняется – выполняется другое действие – выводится сообщение, что год не високосный (if (a = = 0) – условие).

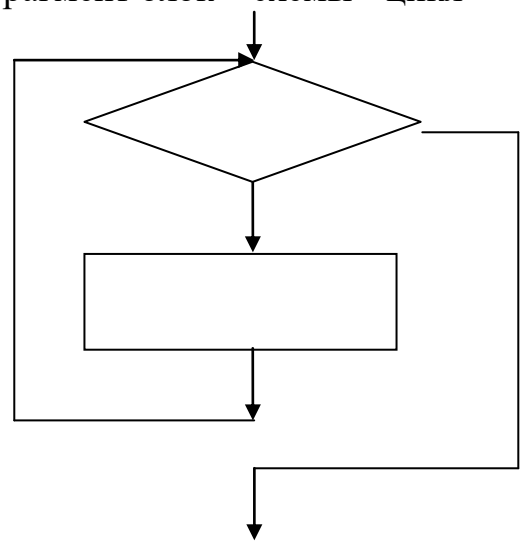
В виде блок - схемы решение задачи можно записать следующим образом:



20. Что представляет собой базовая конструкция цикл?

Цикл – это базовая конструкция, которая задает многократное выполнение оператора.

Фрагмент блок – схемы "цикл"



Пример программы:

Написать программу, выводящую таблицу степеней двойки (до 10-й степени).

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
```

```
int n, x; // n – показатель степени, x – значение 2 в степени n
printf (“Таблица степеней двойки \n”);
x = 1;
for (n = 0; n < = 10; n++)
        {
        printf (“% d % d \n”, n, x);
        x* = 2;
        }
getch ( );
}
```

В этой программе происходит многократное выполнение одного и того же действия – возведение двойки в следующую степень.

Строка "for (n = 0; n < = 10; n++)" означает, что двойка будет возводиться последовательно во все степени от 0 до 10. Когда показатель степени станет больше 10, то выполнение цикла прекратится.

21. Как узнать, что перед нами цикл или, что его необходимо использовать? Что значит цикл с постусловием, с предусловием? Из чего состоит цикл?

Если в программе мы видим, что происходит многократное повторение одного и того же действия, или же нужно это записать для решения задачи (части задачи), то в этой программе присутствует цикл.

Каждый цикл состоит из тела цикла (т.е. операций, которые выполняются несколько раз), начальных установок (что дано изначально), модификации параметра цикла (изменения) и проверки условия продолжения выполнения цикла (когда условие перестает выполняться - цикл заканчивается, и программа переходит к следующему этапу).

Один проход цикла называется итерацией. Если проверка условия выполняется в начале цикла, то это цикл с предусловием, если в конце тела цикла, то цикл с постусловием. Цикл с постусловием обязательно выполняется хотя бы один раз (в отличие от цикла с предусловием).

Параметры цикла – это переменные, изменяющиеся в теле цикла и используемые при проверке условия продолжения. Целочисленные параметры цикла, изменяющиеся с постоянным шагом на каждой итерации – счетчики цикла.

Возможно принудительное завершение текущей итерации и цикла в целом. Для этого используют операторы: break, continue, return, goto.

Пример:

Написать программу, которая проверяет, является ли введенное пользователем число простым.

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
int n, d, r;      // n – число, d – делитель, r – остаток от деления n на d
printf (“Введите целое число: \n”);
scanf (“%d”, &n);
d = 2; //  сначала делим на два
do
        {
```

```
        r = n % d;          тело
        if (r != 0) d++;    цикла
        }
while (r! = 0); // пока n не разделится на d
if (d = = n)
        printf ("% d – простое число! \n", n);
else
        printf ("% d – не простое число! \n", n);
getch ( );
}
```

Этот цикл – цикл с постусловием, т.к. проверка условия происходит после тела цикла. Здесь выполняется несколько раз операция нахождения остатка от деления n на d (d - изменяется с шагом 1 от 2). Следовательно, это и будет телом цикла.

Программа правильно работает для чисел больших единицы. А если n=1, что получится?

22. Как выглядит цикл с предусловием?

Цикл с предусловием имеет вид: while (выражение) оператор, где выражение определяет условие повторения тела цикла. Выполнение оператора начинается с вычисления выражения и, если оно истинно, то выполняется оператор цикла. Если же ложно (при первой проверке равно false), цикл не выполняется ни разу. Выражение повторяется перед каждой итерацией цикла (один проход цикла – итерация).

Пример:

Напечатать таблицу значений функции $f(x) = x^2 + 1$ в заданном диапазоне.

```
#include<stdio.h>
int main (  )
{
float X, Xn, Xk, Dx;
printf ("Введите диапазон и шаг изменения аргумента:");
scanf ("%f %f %f ", &Xn, &Xk, &Dx);
printf (" | X | Y | \n");
X = Xn;   // установка (инициализация) параметра цикла X
while (X < = Xk ) //  проверка условия продолжения
        {
        printf ("|% 5.2 f|% 5.2f| \n", X, X*X+1); // тело цикла
        X+ = Dx;
        }
return 0;
}
```

23. Как выглядит цикл с постусловием?

Цикл с постусловием имеет вид: do оператор while выражение. В этом случае сначала выполняется оператор, а затем вычисляется выражение и, если оно верно, тело цикла выполняется еще раз. Цикл будет завершен, когда выражение станет равным false или когда в теле цикла будет выполнен какой – либо оператор передачи управления.

Пример:

Найти минимальное и максимальное значения среди пяти вводимых натуральных чисел.

```
#include<stdio.h>
#include<conio.h>
int main ( )
{
const int l = 0, r = 32766;
int i, max = 0, min = 32766, n;
for (i = 0, i < 5; i++)
        {
        do
                {
                printf (" n\ Введи число n:");
                scanf ("%d", &n);
                }
        while ((n < l) ||(n  > r));
        if (n > max) max = n;
        if (n < min) min = n;
        }
printf ("Максимальное - %d, минимальное число - %d," max, min);
getch ();
return 0;
}
```

24. Как узнать, что лучше применить while или do while?

Операторы while и do while используются для удобства, а не по необходимости, и все операторы цикла взаимозаменяемы, но:

Оператор do while обычно используют, когда цикл требует обязательно выполнить хотя бы раз (например, если в цикле производится ввод данных).

Пример: Написать программу, которая проверяет, является ли введенное пользователем число четным.

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
const int l = 0, r = 32766;
 int n;
do
        {
        printf ("Введите натуральное число: \n");     тело
        scanf ("%d", &n);                             цикла
        }
while ((n<l)||(n>r));
if (n%2= =0)
        printf ("% d – четное число \n", n);
else
         printf ("% d – нечетное число \n", n);
getch ();
}
```

Оператором while удобнее пользоваться, когда число итераций (повторов) неизвестно, очевидных параметров цикла нет, или модификацию параметров удобно записывать не в конце тела цикла.

Например, при нахождении наибольшего общего делителя

```
void NOD (int a, int b)
{
int c;
while (a > 0)
{
c = b % a;
b = a;
a = c;
}
printf (“NOD = % d”, b);
}
```

здесь число повторов неизвестно, и цикл выполняется до тех пор, пока а больше 0.

Пример:

Написать программу, которая вводит два натуральных числа, сравнивает их и печатает большее число после слова “больше”. Если числа равны, выводится сообщение “эти числа равны”.

```
#include<stdio.h>
#include<conio.h>
int main ( )
{
const int l = 0, r = 32766;
int n, m;
do
        {
        printf (“\n Введи два числа n m:”);
        scanf (“%d %d,” &n, &m);
        }
while ((n < l) || (n > r) || (m < l) || (m > r));
if (n < m)    printf (“Больше % d,” n);
if (m > n)    printf (“Больше % d,” m);
if (m = =n)    printf (“Эти числа равны”);
getch ();
return 0;
}
```

25. Существует ли какой-нибудь вид записи цикла, кроме while и do while?

Да, существует с параметром for, который имеет следующий вид: for (инициализация; выражение; модификация) оператор; Т.е. вначале объявляются переменные и величинам, используемым в цикле, присваивается начальное значение (можно записать несколько операторов, разделенных запятой).

Пример:

```
int k, m;
for (k = 1, m = 0; …;…)…
```

Областью действия переменных, объявленных здесь является цикл.

Далее выражение определяет условие выполнения цикла (если его результат, приведенный к логическому типу bool, равен true, цикл выполняется).

Модификация выполняется после каждой итерации цикла и служит для изменения параметров цикла.

Пример:

Оператор, вычисляющий сумму чисел от 1 до 100:

```
for (int i = 1, s = 0; i < = 100; i++) s+ = i;
```

Пример: нахождение факториала для числа n

```
int fact (int n)
{
int f = 1, i = 1;
for (; i < = n; i++) f* = i;
return f;
}
```

Любая из частей оператора for может быть опущена, но точки с запятой обязательно должны стоять на месте.

26. Что означает фраза, что операторы цикла взаимозаменяемы?

Пример программы, записанной в вопросе 22 с использованием цикла for (программа печатает таблицу значений функции $y = x^2 + 1$ во введенном диапазоне).

С использованием while:

```
#include <stdio.h>
int main ()
{
float X, Xn, Xk, Dx;
printf ("Введите диапазон и шаг изменения аргумента:");");
scanf ("%f %f%f ", &Xn, &Xk, &Dx);
printf (" | X | Y | \n");
X = Xn;   //  установка параметра цикла
while (X < = Xk ) //  проверка условия продолжения
{
printf ("|% 5.2 f|% 5.2f| \n", X, X*X+1); // тело цикла
X+ = Dx;
}
```

С использованием for:

```
#include<stdio.h>
int main ()
{
float X, Xn, Xk, Dx;
prinf ("Введите диапазон и шаг изменения аргумента:");");
scanf ("%f %f %f ", &Xn, &Xk, &Dx);
prinf (" | X | Y | \n");
for (X = Xn;    X < = Xk;  X+ = Dx)
printf ("|% 5.2 f|% 5.2f| \n", X, X*X+1);
return 0;
}
```

Эта программа выполняет те же действия, что и программа из примера для цикла с предусловием, но записана более компактно и наглядно.

Вообще же любой цикл while может быть приведен к эквивалентному циклу for (и наоборот), следующим образом.

```
for (b1; b2; b3)  {составной оператор }
```

```
b1;
while  (b2)  {составной оператор  b3}
```

27. В каких случаях записывается return, в каких его не надо применять?

Оператор return [(выражение)]; возвращает значение или производит возврат из функции в вызвавшую ее функцию.

Функция может содержать несколько операторов return. Если используется тип void, то выражение return не указывается, т.к. эта функция не получает параметры и не возвращает значение (результат). Если тип int, то обязательно return указывается.

Примеры записей:

```
int f1 ( )       { return 1;}    //  правильно, т.к. тип - int
void f2 ( )     { return 1;}     // неправильно,  f2 не должна возвращать
                                                значение, т.к. тип – void.
```

Нельзя возвращать из функции указатель на локальную переменную, т.к. память, выделенная для локальной переменной при входе в функцию, освобождается после возврата из нее.

```
int   *f()
{
int     a = 7;
return &a; // !неправильно, нельзя!
}
```

Пример функции с возвратом значения переменной (формирование числа Фибоначчи с номером n).

```
int Fibonachi (int n)
{
int X = 0, Y = 1,  Z;
while (n > 0)
{
Z = X + Y;
X = Y;
Y = Z;
n --;
return Z;
}
```

Если же значение переменной возвращать не надо, но используется тип int, то return 0;

Пример:

Программа вводит число и определяет четное оно или нечетное.

```
#include<stdio.h>
#include<conio.h>
int main ( )
{
int n;
printf (“Введите натуральное число: \n”);     тело
scanf (“%d”, &n);                                          цикла
if (n%2==0)
        printf (“% d – четное число \n”, n);
else
```

```
        printf ("% d – нечетное число \n", n);
getch ();
return 0;
}
```

28. Что такое goto?

goto- оператор безусловного перехода, один из четырех операторов передачи управления. Он передает управление на "помеченный" оператор, но использовать его крайне нежелательно. Его использование оправдано только в двух случаях:

1. принудительный выход вниз по тексту программы из нескольких вложенных циклов или переключателей.

2. выход из нескольких мест функции в одно.

В остальных случаях использование goto приводит к усложнению программы и затруднению отладки, т.к. он переводит алгоритм из одних условий в другие, а части алгоритма составлены без учета того, что кто-то может войти "не по правилам".

Даже если использование goto оправдано, необходимо произвести некоторые изменения в программе, изменить условия текущего выполнения программы, например, установить начальные (заключительные) значения переменных.

Пример правильного использования goto:

```
retry: for (...){       for (...)
                        {
                        if ( ) goto retry;...   // попытаться сделать все сначала
                        if ( ) goto fatal;
}
}                          // выйти сразу же в конец
fatal;
```

29. Что такое continue, break?

Эти операторы, как и goto и return относятся к операторам, изменяющим естественный порядок выполнения вычислений. Но они более "мягко", по сравнению с goto нарушают логику выполнения программы и рассматриваются, как ограниченные варианты goto.

continue – переход в завершающую часть цикла (он пропускает все операторы, оставшиеся до конца тела цикла, и передает управление на начало следующей итерации).

break – выход из внутреннего цикла. Он используется внутри операторов цикла, if или switsh - для обеспечения перехода в точку программы, находящуюся непосредственно за оператором, внутри которого находится break.

Пример фрагмента конструкции программы:

```
for (i=0; i<n; i++)
 { if (...a[i]...)break;... }
if (i= =n) оператор A;
else оператор B;
```

Т.е. при обнаружении элемента с заданным свойством происходит выход по break. Последняя строка здесь-косвенное определение причин выхода.

То же самое без использования break (с использованием специального признака) выглядит следующим образом:

```
int found, i;
```

```
for (found=0, i=0; ((i < n) && !found); i++)
{ if (…a[i]..) found ++;
…}
if (!found) оператор A;
else оператор B;
```

Т.е. при отсутствии в массиве элемента с заданным свойством выполняется А, иначе действия В.

30. Для чего нужен знак “;”?

Этот символ ограничивает любое выражение, превращая его в оператор. Если этого ограничителя нет, то ошибка обычно выводится компилятором в последующей части программы, когда становится ясно, что это уже не может быть выражением.

Выражение + “;” = оператор

Например:

```
c =  a + b;
```

Символ “;” не всегда ставится в конце каждой строки программы. Например, после выражения с for этот символ не ставится.

```
for (i=1; i<=n; i++)
summ = summ+i;
```

Символ “;” может встречаться в программе и “сам по себе”, т.е. быть пустым оператором. Пустой оператор используется там, где по синтаксису требуется наличие оператора, а по смыслу – нет (никаких действий производить не нужно). Например, в цикле, где все необходимое делается в его заголовке.

```
for (i=0; i<n; i++) s = s+A[i]; //        обычный цикл
for (i=0; A[i]!=0 && i < n; i++); //      цикл с пустым оператором.
```

31. Какой тип данных используется при работе с символами?

Это тип данных char. Размер этого типа – 1 байт, т.е. под каждый символ отводится 1 байт. Этот тип, как и другие целые типы может быть со знаком или без. Диапазон значения этого типа со знаком – от -128 до 127, при использовании спецификатора unsigned диапазон изменяется и становится от 0 до 255. Это достаточно для хранения любого символа из 256 – символьного набора ASC II. Помимо символов величины типа char могут быть целыми числами (если они не превышают границы указанных диапазонов).

Пример программы:

Приветствие.

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
char name [15], fam [20];
printf (“Как Вас зовут?”)’
printf (“Введите имя и фамилию:”);
scanf (“%s”, &name);
scanf (“%s”, &fam);
printf (“Привет, %s %s! \n “, fam, name);
getch ();
}
```

Т.к. имя и фамилия состоят из символов (букв), то при их описании используется тип char, в квадратных скобках указывается максимально возможное количество вводимых символов в имени и фамилии. Т.е. имя и фамилия могут состоять и из меньшего числа символов, но если их количество превысит количество, указанное в скобках, то компилятор не поймет.

32. Каким образом компьютер воспринимает и обрабатывает символы?

Стандартом установлено соответствие между символами и присвоенными им значениями целой переменной. То есть каждому символу соответствует определенный код и любое устройство, отображающее символьные данные, при получении кода выводит соответствующий ему символ. Следовательно, компьютер воспринимает символы в виде кодов.

Например, символу ‘A’ соответствует код 65, а символу ‘.’- код 46 и т.д.

Для обработки символов текста используются те же операции, что и для работы с целыми числами. Нет никаких ограничений на выполнение операций, допустимых для целых переменных (точно так же можно проводить операции сравнения, присваивания, арифметические операции и т.д.).

Пример программы:

Преобразование прописных букв в строчные.

```
# include<stdio.h>
# include<conio.h>
void main ( )
{
unsigned char  st[80];
int i; // номер обрабатываемого символа
puts (“Введите строку:”);
gets (st);
i=0;
while (st[i])
        {
        if ((st[i]>=’a’ && st[i]<=’z’) ||    (st[i] >=’a’ && st[i]<=’п’)) st[i]- = 32;
                else
                if (st[i]>=’p’ && st[i]<=’я’)         st[i]- = 80;
        i++;
        }
puts (st);
getch ();
}
```

В этой программе производятся операции сравнения символов и арифметическая операция вычитания.

33.Каким образом можно преобразовать строчные латинские буквы в прописные?

Это можно сделать как минимум двумя различными способами.

1-ый способ – с помощью библиотечной функции toupper(), которая преобразует строчные символы строки прописные (обрабатывает только буквы латинского алфавита).

Заголовочный файл для этой функции - ctype.h,

прототип: int toupper (int c);

2-ой способ – используя оператор:

if (c >= 'a' && c <= 'z') c = c -'a' + 'A';

Если (код символа ≥ коду буквы a) и (код символа ≤ коду буквы z), то есть попадает в пределы латинского алфавита и, если этот символ – строчная буква, то он преобразуется в "большую", прописную букву.

34. Что общего и в чем различия между функциями printf и puts?

И та и другая функции вывода, но printf() выводит на экран значения переменных разных типов, а puts() выводит только строку символов.

Функция puts() переводит курсор в начало следующей строки экрана.

Если применяется printf(), то для этой цели используется "\n".

И для той, и для другой функций заголовочный файл stdio.h.

Синтаксис puts():

puts (имя_строковой_переменной);

(в качестве параметра функции могут использоваться как строковая константа, так и строковая переменная).

Синтаксис printf():

printf ("строка формата", список выражений);

где в качестве выражений могут быть константы, переменные, бинарные и унарные операции, вызовы функций и т.д.

Пример:

```
…
char st [80]; // введенная строка
puts ("Введите строку:");
gets (st);
…
float a,b,c;
printf ("Введи числа a и  b:\n");
scanf ("%f %f", &a, &b);
printf ("Сумма чисел a + b = %f\n", a + b);
…
```

35. Какие существуют директивы?

Помимо директивы #include, существуют и другие директивы. Например, директива #define, которая определяет подстановку в тексте программ и используется для определения:

а)символических констант

#define имя текст_подстановки (все вхождения имени будут заменены на текст подстановки);

Пример:

define N 25 // значит, что все N будут заменены на 25.

б) макросов (реализующихся подстановкой их текста в текст программы)

define имя (параметры) текст_подстановки

Пример:

define MAX (x,y) ((x)>(y)?(x):(y)) // значит, что везде, где будет появляться имя подстановки (MAX (x,y)) будет происходить условная операция ((x)>(y)?(x):(y));

в) символов, управляющих условной компиляцией (они используются вместе с директивами #ifdef и #ifndef).

#undef имя.

Существуют также директивы # if, # ifdef , # infdef и #elif – директивы условной компиляции. Они применяются для того, чтобы исключить компиляцию других частей программы.

Также, существует директива #undef, которая удаляет определение символа. Она используется редко, например, при отключении какой-либо опции компилятора.

36. Что такое switch?

switch, так же как и if, - инструкция выбора. Но если оператор if предназначен для выбора одного из двух возможных направлений дальнейшего хода программы, то switch – для выбора одного из нескольких направлений. Выбор последовательности действий осуществляется в зависимости от равенства значений переменной – селектора-константы, указанной после слова case. Если значение не равно ни одной из констант, то выполняются действия расположенные после слова default. В качестве переменной – селектора можно использовать переменную типа int либо char.

Пример:

Написать программу, которая запрашивает у пользователя номер дня недели, затем выводит название дня недели или сообщение об ошибке, если введены неверные данные.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int nd;  // номер дня недели
puts (“Введите номер дня недели (1-7):\n”);
scanf ("%d”, &nd);
switch (nd)
{
        case 1: puts (“Monday”);    break;
        case 2: puts (“Tuesday”); break;
        case 3: puts (“Wednesday”); break;
        case 4: puts (“Thursday”); break;
        case 5: puts (“Friday”); break;
        case 6: puts (“Saturday”); break;
        case 7: puts (“Sunday”); break;
        default: puts (“В неделе – 7 дней! \n”);
}
getch ();
}
```

37. Что такое рекурсия?

Рекурсивной функцией называется функция, которая вызывает саму себя. Такая рекурсия называется прямой. Существует так же косвенная рекурсия – когда две или более функций вызывают друг друга.

Существует линейная рекурсия (определение объекта включает в себя единственный объект) и ветвящаяся рекурсия (таких включаемых объектов много).

Если функция вызывает себя, то создается копия значений ее параметров, а затем управление передается первому исполняемому оператору функции. При повторном вызове процесс повторяется.

Рекурсия должна иметь внутри себя условие завершения, по которому очередной шаг ее становится невозможным, иначе она будет бесконечной.

Классическим примером рекурсии можно назвать вычисление факториала.

```
int fact_rec (int n)
{
if (n = = 0 || n = = 1) return 1;
return  (n* fact_rec (n – 1 ));
}
```

Любую рекурсивную функцию можно реализовать без применения рекурсии.

Пример: программа, записанная выше, но без применения рекурсии.

```
int fact (int n)
{
int f = 1;
for (i = 1; i < = n; i++)
        f* =i;
return f;
}
```

“+” рекурсии – компактная запись.

“-“ рекурсии – расход времени и памяти на повторные вызовы функции и передачу ей копий параметров.

38. Как поступают, если в программе необходимо работать не с одним объектом, а с последовательностью объектов?

Для таких случаев существуют массивы.

Массив – это набор объектов одного типа, имеющих одно имя, располагающихся в последовательных ячейках памяти.

Внешне описание массива в программе отличается от описания простой переменной наличием после имени квадратных скобок, в которых задается количество элементов массива (размерность).

Размерность массива, вместе с типом его элементов определяет объем памяти, необходимый для размещения массива, поэтому размерность может быть только целым числом.

Пример:

```
float Array [15]; // описание массива, состоящего из 15 вещественных элементов.
```

Элементы массива нумеруются с нуля.

Инициализирующие значения для массивов записываются в фигурных скобках.

Если элементов в массиве больше, чем инициализаторов, элементы, для которых значения не указаны, обнуляются.

Пример:

```
int a[5]={5,4,7}; // a[0]=5, a[1]=4, a[2]=7, a[3]=0, a[4]=0
```

Для доступа к элементам массива после его имени указывается номер элемента (индекс) в квадратных скобках.

Пример (фрагмент программы вычисления суммы элементов массива):

```
for (i=0, sum=0; i < n; i++) sum+ = a[i];
```

здесь i - номер элемента массива, происходит последовательный перебор всех элементов массива.

Идентификатор массива является константным указателем на его нулевой элемент. Например, имя a из первого примера это то же самое, что и &a[0].

39. Как можно подсчитать количество положительных элементов массива?

В этом случае, как и в любом другом, где требуется подсчитать количество элементов, удовлетворяющих тому или иному условию, используется так называемая переменная-счетчик, суть которой сводится к записи вида:

```
for (i=0, K=0; i < n; i++)
        if (Array [i] > 0) K++;
```

где Array – название массива, i - его элементы. Т.е. если элемент массива больше 0, то к К прибавляется единица (операция инкремента) и так перебирается каждый элемент массива.

40. Каким образом можно осуществить подсчет суммы положительных элементов массива?

Сделать это поможет переменная-накопитель, запись фрагмента программы, где необходимо накопить что либо, сводится к:

for (s = 0;…;…) { получить s = s + k;}

Т.е. переменная s накапливает сумму значений k, полученных на каждом из шагов выполненного цикла (на каждом шаге к значению переменной s добавляется новое k и в результате запоминается в том же самом s).

Для данной задачи фрагмент записи будет выглядеть следующим образом:

```
for (s = 0, i  = 0; i < n; i++)
        if  (A [i] > 0)   s + = A[i];
```

где s - переменная накопитель, n - размер массива, i - элемент массива, если элемент больше 0, то переменной-накопителю (с начальным значением 0) прибавляется этот элемент.

41. Как найти номер максимального элемента в массиве?

Здесь используется классическая, слегка видоизмененная схема нахождения max (min) элемента, который в общем виде выглядит так:

```
for (s= меньше меньшего (больше большего);…;…)
        {получить k, if (k > s) s = k},
```

т.е. если новое значение больше, чем то, которое имеется – оно запоминается, иначе оставляем старое.

Типичный пример – нахождение max элемента массива

```
for (s = A [0], i= 1; i < 10; i++)          if (A[i] > s) s = A[i];
```

В нашем же случае эта конструкция будет видоизменена и получим:

```
for (i= 1, k = 0; i < 10; i++)      if (A[i] > A[k]) k = i;
```

Вначале полагаем, что номер max-го элемента массива равен 0, затем на каждом шаге сравниваем все последующие элементы с max-ным и если необходимо меняем значение индекса max.

Запоминается не само значение максимума, а номер элемента в массиве, где оно находится.

42. Что называется динамической памятью, динамическим массивом?

Динамическая память – это свободная память, в которой во время выполнения программы можно выделить место в соответствии с потребностями.

Массивы, создаваемые в динамической памяти, называются динамическими. Они создаются с помощью операции выделения динамической памяти new и функции malloc().

Пример (с использованием операции new)

```
int   n = 100;
float *p = new float [n];
```

здесь создается переменная-указатель на float. В динамической памяти отводится область достаточная для размещения 100 элементов этого типа, адрес начала этой области записывается в p.

Пример (с помощью функции malloc())

```
int n = 100;
float * q = (float*)malloc (n* sizeof (float));
```

В первом случае память освобождается операцией delete (delete [ ]p;)

во втором – с помощью функции free() (free (q);).

Динамические массивы, в отличие от обычных, нельзя инициализировать при описании, и они не обнуляются автоматически. Их преимущество в том, что их размерность может быть переменной. Доступ к элементам динамического массива осуществляется так же, как и доступ к элементам статического массива (например, p[5] или *(p+5) .

Пример:

```
// динамический массив заданной размерности
int *Get_Array (int &n)
{
int n, i, *p;  // n – размерность массива, *p – указатель на тип int
printf ("Введи количество элементов в массиве:");
scanf ("%d", &n);
if ((p =  new int [n]) = =NULL)   { puts("Нет памяти!"); return NULL;}
for (i  = 0; i < n; i++)
{
        printf ("Введи %d - й элемент:",i);
        scanf ("%d",  &p[i]);
}
return p; // вернуть указатель на начало массива
}
```

43. Каким образом можно написать программу, вычисляющую определитель квадратной матрицы второго порядка? Каким образом вообще можно работать с матрицами?

Для работы с матрицами пользуются многомерными массивами, которые задаются указанием каждого измерения в квадратных скобках. Например, запись int [2][2] означает, что в массиве 2 столбца и 2 строки.

В памяти такой массив располагается построчно, т.к. память одномерна.

Для доступа к элементу многомерного массива указывают все его индексы, например, mart [i][j].

При инициализации многомерного массива он представляется следующим образом:

```
int mass 2[ ] [ ] = {{5,5},{7,7}};
```
либо
```
int mass [2][2] = {5,5,7,7};
```
Программа для решения нашей задачи может выглядеть следующим образом
```
#include<stdio.h>
#include<conio.h>
void main ( )
{
float a [2][2];
float opr;      // определитель
int  i, j;      // индексы массива
puts ("Введите матрицу второго порядка");
for (i = 0; i < 2; i++)
        {
        printf  ( "→");
        scanf ("%f  %f", &a[i][0], &a[i][1]);
}
opr = a [0][0]* a [1][1] – a [0][1] * a [1][ 0];
puts  ("Определитель матрицы ");
for (i = 0; i < 2; i++)
        printf ("%f \t %f \n", a [i][0], a  [i][1]);
printf ("равен %f ", opr);
getch ();
}
```

44. Как можно поменять местами элементы массива, чтобы оказалось, что последующий элемент больше предыдущего?

Для перестановки местами двух переменных требуется третья.

Наглядный пример: для того, чтобы обменять содержимое двух стаканов (без смешивания), необходим третий. Это правило трех стаканов.

Пример: Обмен значений переменных a и b с использованием переменной c ("правило трех стаканов").

```
int a = 5, b = 6, c;
c = a; // перелить содержимое первого стакана в пустой (третий)
a = b; // перелить второй в первый
b = c; //  перелить третий во второй
```
Наша задача будет выглядеть так:
```
void F (int A[ ], int n)
{
int cc, i, found;
do
{
        found = 0;
        for (i =  0;  i < n – 1; i++)
                if (A [i] >A [i+1])
                        {
                         cc = A[i]; A[i] = A[i+1]; A[i+1] = cc;
                         found ++;
                        }
} while (found ! = 0);
}
```

45. Как найти номер максимального элемента в массиве?

Здесь используется классическая, слегка видоизмененная схема нахождения max (min) элемента, который в общем виде выглядит так:

for (s = меньше меньшего (больше большего); …; …)

{получить k, если (k > s) s=k}, т.е. если новое значение больше, чем то, которое имеется – оно запоминается, иначе – оставляется старое.

Типичный пример – нахождение max элемента массива

```
for (s = A [0], i= 1; i < 10; i++) if (A[i] > s) s = A[i];
```

В нашем же случае эта конструкция будет видоизменена и получим:

```
for (i = 1, k = 0; i < 10; i++) if (A[i] > A[k]) k = i;
```

т.е. запоминается не само значение максимума, а номер элемента в массиве, где оно находится.

46. В каком порядке выполняются операции в выражении? Можно ли изменить этот порядок?

Операции выполняются в соответствии с их приоритетами. Так, приоритет унарных операций выше, чем у бинарных и тернарных. Т.е. если в одном выражении величина умножается на другую величину и увеличивается на один (операция инкремента), то вначале выполнится операция ++ и только потом числа будут перемножены.

Также приоритеты различаются и внутри этих групп. Т.е. приоритеты бинарных (унарных, тернарных) операций также могут быть различны. Например, приоритет операции умножения выше, чем у сложения. Т.е. в выражении 2+2*2 (1) вначале произойдет умножение (т.е. получится 4),а затем к четырем прибавится 2, т.е. результатом этого выражения будет 6.

Когда в выражении записано несколько операций одинакового приоритета, то унарные операции, условная операция и операция присваивания выполняются справа налево, остальные – слева направо. Например, a=b=c; означает a=(b=c), а a+b+c означает (a+b)+c.

Изменить порядок выполнения операций можно, для этого нужно использовать круглые скобки. Т.е. результат выражения (1) изменится, если поставить круглые скобки. (2+2)*2 в ответе даст уже 8, т.к. вначале выполнится сложение в скобках, а только затем умножение.

47. Какие операции относятся к логическим? Для чего они нужны?

К этим операциям относятся такие операции, как операция И (&&) и операция ИЛИ(||). Операнды (то, над чем производятся действия) этих операций могут иметь арифметический тип или быть указателями.

Операнды в каждой операции могут быть различных типов, преобразования типов не производятся, (каждый оператор рассматривается с точки зрения его эквивалентности нулю).

Результатом логической операции является true (истина) или false (ложь). При этом результат операции логическое "И" имеет значение true, только если оба операнда имеют значение true. Результат же операции логическое "ИЛИ" имеет значение true, если хотя бы один из операндов имеет значение true.

Пример программы с использованием логического "ИЛИ".

Написать программу, которая считывает радиус круга и печатает диаметр и площадь круга.

```
#include<stdio.h>
#include<conio.h>
int main ()
{
const float pi=31415926, l=0.1, r=(3.4E+38)-1;
float radius;
do
{
printf ("\n Enter radius");
scanf  ("%f",    &radius);
}
while ((radius < l) || (radius > r));
printf ("D=%f, S=%f \n ", 2*radius, pi* radius);
getch ();
return 0;
}
```

Значение радиуса в этой программе можно вводить до тех пор, пока оно будет меньше наименьшего из возможных значений (0.1) или больше наибольшего (3.4E+38)-1. Если оба условия станут ложными, то программа идет дальше.

Пример программы с логическим "И":

Определить имеют ли одинаковую четность три натуральных числа.

```
#include<stdio.h>
#include<conio.h>
void Chetnost (unsigned int &a, unsigned int &b, unsigned int &c)
{
printf ("\n Введите три числа a b c: ");
scanf  ("%u %u %u", &a, &b, &c);
if ((a%2 ! = 0) && (b%2 != 0))
if (c%2 != 0) printf ("Yes");
else
{if ((a%2 == 0) && (b%2 == 0))
if (c%2 ==0) printf ("Yes");
else              printf ("No"  );
}
}
void main ( );
{
unsigned int a, b, c;
clrscr ( );
Chetnost (a,b,c);
…
getch ( );
}
```

Для того чтобы числа имели одинаковую четность (т.е. были все четны или все нечетны) необходимо, чтобы все три числа при делении на 2 давали в остатке 0 либо все давали отличный от нуля остаток. Т.е. когда остатки у a и b одновременно отличны от нуля и, если c дает отличный от нуля остаток, то ответ “Yes” (числа имеют одинаковую четность).

48. Для чего нужно разделение программ по функциям?

Функция – это именованная последовательность описаний и операторов, выполняющая какое либо законченное действие. Функция может принимать параметры и возвращать значение.

Разделение на функции существенно упрощает чтение программы. Ведь с увеличением объема программы становится невозможным удерживать все ее детали в памяти. А разбиение программы на более простые и обозримые части помогает бороться с этим.

Кроме того, разделение программы на функции позволяет избежать избыточности кода. Происходит это благодаря тому, что функция записывается один раз, а вызывать ее на выполнение можно неоднократно, причем из разных точек программы. Процесс отладки программы, содержащей функции, можно лучше структурировать.

Пример программы с использованием функции:

Идет k-я секунда суток. Определить сколько полных часов (h) и полных минут (m) прошло к этому моменту.

```
#include<stdio.h>
#include<conio.h>
void Time (int &h, int &m, long &sec)
{
do{
printf (“Enter k (0..86400): “);
scanf ( “%ld”, &sec);
}while ((sec < 0) && (sec > 86400));
h = sec / 3600;      m = (sec % 3600) / 60;
printf (“hour = %d, min = %d sec = %d”, h, m, (sec-h*3600-m*60));
}
void main ()
{
int h, m;
long k;
Time (h, m, k);
}
```

Часто используемые функции можно помещать в библиотеки. Это позволяет создавать более простые в отладке и сопровождении программы.

49. Для чего нужны сортировки, какими они бывают? Что такое ключ?

Для того, чтобы найти что-либо интересующее нас желательно, чтобы данные были упорядочены, иначе придется осуществлять последовательный перебор всех элементов (а элементами массива могут являться более сложные образования, чем простые переменные). Для того чтобы проще и удобнее находить нужный элемент и существуют различные виды сортировок.

Ключ – это часть элемента данных, которая используется для его идентификации и поиска среди множества других таких элементов.

Сортировки различают:

а) по виду размещения элементов: внутренняя сортировка – в памяти и внешняя – в файле данных;

б) по виду структуры данных, содержащих сортируемые элементы: сортировка массивов, массивов указателей и т.д.;

в) основными идеями алгоритма сортировки.

Различают:

сортировки вставками – очередной элемент помещается по месту своего расположения в выходную последовательность (массив).

Пример простой вставки:

```
void sort(int in[], int n)
{
int i, j, k, v;
for (i = 1; i < n; i++)
        {
        v = in[i];
        for (k = 0; k < i; k++)      if (in[k] > v ) break;
        for (j = i - 1; j > = k; j--)  in[j + 1] = in[j];
        in[k] = v;
        }
}
```

сортировку выбором – выбирается очередной минимальный элемент и помещается в конец последовательности.

Пример сортировки выбором:

```
void sort(int in[ ], int n)
{
int i, j, k, c;
for (i = 0; i < n – 1; i++)
        {
        for (j = i + 1, k = i; j < n; j++)  if (in[j] < in[k])      k = j;
        c = in[k];      in[k] = in[i];          in[i] = c;
        }
}
```

сортировку разделением – последовательность (массив) разделяется на две частично упорядоченные по принципу “больше-меньше” части, которые затем могут быть отсортированы независимо (в том числе тем же самым алгоритмом).

Схема сортировки рекурсивным разделением:

```
void sort (int in[], int a, int b)
{
int i;
if (a >= b) return;
// Разделить массив в интервале a…b на две части a…i -1 и i..b
// относительно значения v по принципу <v, >=v
sort (in, a, i-1);
sort (in, i,  b);
}
```

сортировку слиянием – последовательность регулярно распределяется на несколько независимых частей, которые затем объединяются (слияние).

обменные сортировки – группа сортировок с многочисленными оптимизациями, основанная на регулярном обмене соседних элементов.

сортировка подсчетом – сортировка, в которой определяется количество элементов, больших или меньших данного, и определяется его расположение в выходном массиве.

Пример сортировки подсчетом (неполной):

```
void sort(int in[ ], int out[ ], int n)
```

```
{
int i, j, cnt;
for (i = 0; i < n; i++)
{
for ( cnt = 0, j = 0; j < n; j++)
if (in[j] < in[i]) cnt++;      // счетчик элементов больше текущего
out [cnt] = in[i];             // определяет его место в выходном массиве
}
}
```

50. Что такое сортировка методом “пузырька”?

Эта сортировка относится к сортировкам обменом. Суть ее заключается в том, что элементы сравниваются попарно и, если пара расположена не в порядке возрастания, производится перестановка. Этот процесс повторяется до тех пор, пока при просмотре массива от начала до конца перестановок больше не будет.

Пример:

```
void sort (int A[], int n)
{
int cc, i, found;
do{
        for (i=0, found = 0; i < n-1; i++)
        if (A[i]>A[i+1])
                { cc=A[i]; A[i]=A[i+1]; A[i+1]=cc;
                  found ++; }
    } while (found != 0);
}
```

В этой функции found - количество сравнений, i- номер элемента. Вначале считаем количество сравнений равным нулю. Производим последовательный перебор всех элементов массива от первого, до предпоследнего. Сравниваем соседние элементы и, если предыдущий элемент больше последующего, то производится перестановка.

Дополните **определения (**в заданиях 1 - 20**)**

1. Существует множество управляющих базовых структур: *цепочка*, *ветвление*, *цикл* и *подпрограмма*. Алгоритм любой сложности можно описать, используя ограниченные подмножества структур:

а) _____ и цепочка;
б) ветвление и _____.

2. Эффективность — показатель, характеризующий _____ и объем используемой _____.

3. Выполнение программы на языке Си начинается с функции _____.

4. Каждый оператор заканчивается _____ _ _____.

5. Любой элемент данных относится к _____ типу, явно указываемому в описании _____, переменной или функции.

6. Объем памяти, отводимый для переменной, определяется в соответствии с _____ _____, которые может принимать переменная.

7. Переменная, которая известна только внутри функции, в которой она определена, называется _____.

8. При передаче параметров в функцию аргументы типа float всегда преобразуются к типу _____.

9. Оператор _____ в вызываемой функции передает управление и возможно значение в вызывающую функцию.

10. Ключевое слово _____ используется в заголовке функции, чтобы указать, что функция не возвращает значение или указать, что она не содержит параметров.

11. _____ функции позволяет компилятору проверить количество, типы и порядок следования аргументов, передаваемых функции.

12. Переменная, объявленная вне определений функций, является _____ переменной.

13. Функция не может вернуть массив в качестве результата, но может вернуть _____ на массив.

14. Агрегирование — _____ данных, при котором не обязательно наличие какой бы то ни было _____ взаимосвязи данных.

15. Массив — структура, состоящая из _____ количества компонент _____ типа.

16. Элементы массива занимают _____ ячейки памяти.

17. Имя массива есть _____ указатель на его начало.

18. Структурой данных называют набор из одного или нескольких имен и _____ данных, к которым эти имена позволяют получить _____.

19. Массив и структура обеспечивают _____ доступ к элементам.

20. Для структур данных предусмотрены операции _____ компонент и соответствующая _____.

Выберите все *верные* утверждения (в заданиях 21 - 23)(назовите буквы)

21. *В языке Си существуют пять типов операторов:*

A. Сравнения
B. Повторения
C. Присваивания
D. Передачи данных
E. Передачи управления
F. Возврата значения
G. Передачи параметра
H. Вывода результата
I. Вызова компилятора
J. Вызова функции
K. Пустой оператор
L. Отношения
M. Описания
N. Проверки

Ответ __,__,__,__,__.

22. *Запятая разделяет:*

A. Аргументы в списках формальных и фактических параметров
B. Переменные разного типа в операторах присваивания
C. Начальные значения элементов массива в операторах описания
D. Размерности массивов в операторах описания
E. Переменные одного типа в операторах описания
F. Список возвращаемых значений в операторе return
G. Операторы присваивания в заголовке цикла for

Ответ _____.

23. *Аргумент, передаваемый в функцию, может быть:*

A. Константой
B. Операндом
C. Строковым литералом
D. Переменной
E. Выражением
F. Операцией

Ответ _____.

24. Выберите все *верные* варианты ответов: «да» или «нет»

Оператор вызова функции - имя функции, за которым в круглых скобках следует список формальных параметров.	да / нет
Заголовок функции – это часть определения функции, ограниченная круглыми скобками.	*да / нет*
Определение функции – это заголовок и тело функции.	да / нет
Массив всегда передается в функцию через указатель на его начало.	*да/нет*
Массив может возвращаться как результат работы функции.	да / нет
Тип возвращаемого функцией значения всегда void.	*да / нет*
Объявление функции – это заголовок и возвращаемое функцией значение.	да / нет

Ответ ___,___,___,___,___,___,___.

Установите соответствие (в заданиях 25 - 28**)(**назовите буквы**)**

25.

Тип данных	Объем памяти	Диапазон значений
1) char **2)** *unsigned char* **3)** int **4)** *unsigned int* **5)** long **6)** *unsigned long* **7)** float **8)** *double* **9)** long double	**A.** 2 байта **B.** 4 байта **C.** 8 байтов **D.** 1 байт **E.** 10 байтов	**a)** 3.4E-38…3.4E+38; **b)** *0… 65535*; **c)** 0… 4294967295; **d)** *-32768…+32767*; **e)** 3.4E-4932…1.1E+4932; **f)** *-2147483648… 2147483647*; **g)** -128… 127; **h)** *1.7E-308…1.7E+308*; **i)** 0…255

Ответ 1)__,__ 2)__,__ 3)__,__ 4)__,__ 5)__,__ 6)__,__ 7)__,__ 8)__,__ 9)__,__.

26.

Использование скобок	**Типы скобок**
1. индексирование элемента массива 2. условное выражение в операторе if 3. явное преобразование типа 4. определение, объявление и вызов функции 5. выделяют тело функции или составной блок 6. в формальных параметрах при передаче многомерных массивов 7. *переключающее_ выражение* оператора switch 8. *список_константных_выражений* оператора switch 9. инициализация скалярных переменных в операторах описания 10. выделение списка компонентов в определении структуры 11. в макроопределениях, обрабатываемых препроцессором 12. обязательные элементы в операторы циклов 13. инициализация массивов в операторах описания 14. изменение последовательность выполнения операций 15. при описании массива	**1) ()** **2) []** **3) {}**

Ответ 1)__,__,__,__,__,__,__,__ 2)__,__ ,__ 3)__,__,__,__.

27.

Объекты программы	**Выражения**
A. Имя массива **B.** Ссылки на объекты **C.** Имя функции **D.** Имена скалярных арифметических и символьных переменных **E.** Имена переменных, принадлежащих массивам **F.** Вызов функции **G.** Выражения с операцией разыменования ‘*’ **H.** Имена указателей **I.** Имя константы	**1)** Леводопустимые **2)** Праводопустимые

Ответ 1)__,__,__,__,__ 2)__,__,__,__.

28.

Объекты	**Типы данных**
A. Целые переменные **B.** Массивы **C.** Классы **D.** Перечисления **E.** Вещественные переменные **F.** Объединения **G.** Ссылки **H.** Указатели **I.** Структуры	**1)** Агрегатные **2)** Скалярные

Ответ 1)__,__,__,__,__ 2)__,__,__ ,__.

Выберите *правильный* ответ (в заданиях 29 - 30)(назовите буквы)

29.

Программа, фрагмент программы на языке С	Результат работы

1. void func (int a, int *b)
{
a=3; *b=4;
}
void main()
{
int a=1,b=2;
func(a, &b);
printf("%d %d", a, b);
}
	A. 1 2
	B. 3 4
	C. 3 2
	D. 1 4

2. void main()
{
char* s="simple string", *p=s;
int i=0;
while (*p++) i++;
printf("%d", i – strlen (s));
}
	A. –1
	B. 1
	C. 0
	D. ничего

3. char* s=”\1\1”; short int * p= (short int*)s; printf(“%d”,*p);	**A.** 1 1 **B.** \1\1 **C.** \\1\\1 **D.** ничего

Ответ 1__ 2__ 3__.

30. Оператору **w = v;** эквивалентна следующая последовательность операторов

A. p = &v; w = *p;

B. p = &w; w=*p;

C. v = &w; w=*v;

Ответ ___.