МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

А.А. Белеванцев, С.С. Гайсарян, В.П. Иванников,
Л.С. Корухова, В.А. Падарян

ЗАДАЧИ ЭКЗАМЕНОВ ПО ВВОДНОМУ КУРСУ ПРОГРАММИРОВАНИЯ

(учебно-методическое пособие)

Москва
2012

УДК 004.02+004.043(075.8)
ББК 32.973-018я73
З-15

Печатается по решению Редакционно-издательского Совета факультета вычислительной математики и кибернетики МГУ имени М.В. Ломоносова

Рецензенты:
С.Ю. Соловьев, профессор
А.К. Терехин, доцент

А.А. Белеванцев, С.С. Гайсарян, В.П. Иванников, Л.С. Корухова, В.А. Падарян. Задачи экзаменов по вводному курсу программирования (учебно-методическое пособие). — М. Издательский отдел факультета вычислительной математики и кибернетики МГУ (лицензия ИД № 05899 от 24.09.2001), 2012. – 68 с.

ISBN 978-5-89407-497-9
ISBN 978-5-317-04300-1

Пособие содержит варианты коллоквиумов и письменных экзаменов основных лекционных курсов по программированию, читаемых для студентов 1 потока 1 курса факультета ВМК МГУ. Варианты письменного экзамена включают задачи различного уровня сложности. В пособии даны ответы, комментарии и указания к решению, а также продемонстрированы авторские решения некоторых задач.

Методическое пособие предназначено для преподавателей, ведущих практические занятия в поддержку лекционных курсов по программированию для студентов 1 курса, а также рекомендуется студентам при подготовке к экзамену.

ISBN 978-5-89407-497-9
ISBN 978-5-317-04300-1

Введение

Пособие содержит варианты коллоквиумов и письменных экзаменов основных лекционных курсов по программированию (“Алгоритмы и алгоритмические языки” и “Архитектура ЭВМ и язык ассемблера”), читаемых для студентов 1 потока 1 курса факультета ВМК МГУ. Варианты письменного экзамена включают задачи различного уровня сложности: наряду с простейшими задачами и вопросами по конкретной теме, имеются задачи, предполагающие знание материала нескольких тем и умение владеть этим материалом. Задачи коллоквиумов, как правило, преследуют цель – проверить усвоение студентом конкретных тем лекционного курса. При проведении письменного экзамена авторы составляли задачи и варианты, стараясь включить в них все основные темы курса.

В первом разделе пособия приведены экзаменационные вопросы по курсам и рекомендуемая литература. В третьем разделе для приведенных типовых задач экзаменов и коллоквиумов даны ответы, комментарии и указания к решению, а также продемонстрированы авторские решения некоторых задач,

1. ЭКЗАМЕНАЦИОННЫЕ ВОПРОСЫ ПО КУРСУ

1.1. КУРС ЛЕКЦИЙ "АЛГОРИТМЫ И АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ" (2011/2012 учебный год, осенний семестр)

I. ***Формальные системы описания алгоритмов.***

1. Задачи обработки информации и алгоритмы. Неформальное (интуитивное) определение алгоритма.
2. Формализация алгоритма. Машина Тьюринга.
3. Способы представления машин Тьюринга. Нормальные вычисления.
4. Диаграммы Тьюринга. Построение диаграмм Тьюринга. Построение таблиц по диаграммам.
5. Понятие универсальной машины Тьюринга. Построение универсальной машины Тьюринга.
6. Проблема останова и алгоритмическая неразрешимость.
7. Алгоритмическая неразрешимость проблемы самоприменимости.
8. Тезис Тьюринга – Чёрча.
9. Нормальные алгоритмы Маркова. Эквивалентность формальных систем описания алгоритмов.

II. ***Язык программирования Си.***

1. Язык программирования Си. Описание Си-машины.
2. Структура Си-программы.
3. Типы данных. Переменные, константы. Классы памяти. Области видимости и существования переменных.
4. Арифметические типы данных. Арифметические операции над целочисленными данными.

5. Приведение типов. Правила неявного преобразования типов («обычные арифметические преобразования»).
6. Старшинство операций.
7. Операторы языка Си: условные операторы, операторы цикла, операторы перехода, составной оператор (блок).
8. Символьный тип данных (char).
9. Массивы и строки.
10. Указатели. Адресная арифметика. Преобразование типа указателя.
11. Указатели и массивы. Массивы указателей. Многомерные массивы.
12. Функции. Объявление функции. Формальные параметры. Возвращаемое значение. Побочный эффект функции. Функции типа void.
13. Определение функции. Библиотечные функции. Вызов функции. Фактические параметры и способ их передачи (по значению). Указатели и параметры функции. Передача массива в функцию. Квалификатор restrict.
14. Рекурсивные функции. Квалификатор inline.
15. Операция sizeof.
16. Арифметические операции над данными с плавающей точкой.
17. Отношения и логические операции. Поразрядные операции. Реализация абстрактного типа данных «множество».
18. Структуры, перечисления, объединения. Указатели на структуры. Битовые поля.
19. Динамическое распределение памяти.
20. Указатель на функцию.
21. Сборка Си-программы: препроцессирование, компиляция, компоновка. Директивы препроцессора. Директива #include и заголовочные файлы. Условная компиляция.
22. Стандартные библиотеки языка Си.

III. ***Алгоритмы и структуры данных.***

1. Динамические структуры данных. Список (однонаправленный и двунаправленный). Стек и его реализация на массиве и на списке. Очередь.
2. Применение стека для преобразования выражений в польскую запись.
3. Топологическая сортировка узлов ациклического ориентированного графа: постановка задачи, алгоритм Вирта.
4. Сортировка. Основные алгоритмы сортировки. Оценка сложности алгоритмов сортировки.
5. Быстрая сортировка Хоара.
6. Двоичное дерево. Представление двоичного дерева в памяти компьютера.
7. Способы обхода двоичного дерева и их рекурсивная и нерекурсивная реализации.
8. Прошитое двоичное дерево. Прошитое двоичное дерево с заголовком.
9. Двоичные деревья поиска. Реализация операций поиска элемента (*search);* вставки элемента (*insert*) и удаления элемента (*delete*). Реализация операций *min* (минимум), *max* (максимум), *pred* (предыдущий) и *succ* (следующий).
10. Построение двоичного дерева поиска.
11. Структура данных «пирамида». Пирамидальная сортировка.
12. Сбалансированные двоичные деревья. Деревья Фибоначчи. Число узлов в дереве Фибоначчи высоты *h.*
13. АВЛ-деревья. Базовые операции над АВЛ-деревьями и их реализация.
14. Балансирование АВЛ-деревьев. Реализация операции вставки узла в АВЛ-дерево. Построение АВЛ-дерева.

15. Красно-черные деревья. Вставка узла в красно-черное дерево.
16. Словарные операции и их реализация с помощью хеш-функций. Методы построения хеш-функций.
17. Хеширование с цепочками. Хеширование с открытой адресацией. Двойное хеширование.
18. Оценка среднего времени успешного поиска в хеш-таблице с заданным коэффициентом заполнения.
19. Цифровой поиск.
20. Поиск подстрок по образцу. Алгоритм Кнута – Морриса – Пратта.
21. Алгоритмы перебора множеств. Рекурсивный алгоритм генерации перестановок. Алгоритм Нарайаны.

IV. *Рекомендованная литература*

1. Л.С. Корухова, М.Р. Шура-Бура. Введение в алгоритмы. (учебное пособие для студентов 1 курса) - М., Издательский отдел факультета ВМК МГУ, 2009.
2. В.П. Иванников, Л.С. Корухова, В.Н. Пильщиков. Курс “ «Алгоритмы и алгоритмические языки» Варианты письменного экзамена. М., Издательский отдел факультета ВМК МГУ, 2007.
3. М. Минский. Вычисления и автоматы. - М., “Мир”, 1971.Г.
4. Эббинхауз, К. Якобс, Ф. Манн, Г. Хермес. Машины Тьюринга и рекурсивные функции. М., “Мир”, 1972.
5 Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. Алгоритмы: построение и анализ. – 2-е изд. М., “Вильямс”, 2011...
6. Б. Керниган, Д. Ритчи. Язык программирования Си. 2-е изд. М., “Вильямс”, 2010
7. Г. Шилдт. Полный справочник по Си. 4-е изд, М., “Вильямс”, 2010.
8. Стандарт языка Си C99 + TC{1,2,3}. Доступен с сайта

http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1256.pdf
8. Н. Вирт. Алгоритмы + структуры данных = программы. - М., “Мир”, 1985.
9. Д. Кнут. Искусство программирования для ЭВМ.
Т.1. Основные алгоритмы.
Т.2. Сортировка и поиск.
- М., изд-во Вильямс, 2010, 2011, 2012.
10. А. Ахо, Д. Хопкрофт, Д. Ульман. Структуры данных и алгоритмы. - М., изд-во Вильямс, 2000.
11. А.А. Белеванцев, С.С. Гайсарян, Л.С. Корухова, Е.А. Кузьменкова, В.С. Махнычев. Семинары по курсу «Алгоритмы и алгоритмические языки» (учебно-методическое пособие для студентов 1 курса) - М., Издательский отдел факультета ВМК МГУ, 77 с., 2012.

1.2. КУРС ЛЕКЦИЙ "АРХИТЕКТУРА ЭВМ И ЯЗЫК АССЕМБЛЕРА” (2011/2012 учебный год, весенний семестр)

I. ***Базовые понятия архитектуры ЭВМ***

1. Архитектура фон Неймана. Основные компоненты компьютера. Машинная команда, типы операндов, адресность ЭВМ, такт работы. Упрощенная схема выполнения инструкции.
2. Упрощенная схема компиляции Си-программы в окружении: компилятор gcc, операционная система Linux, архитектура IA32.
3. Архитектура IA32: основные регистры, форматы команд. Представление машинной команды, ассемблерная инструкция.
4. Организация ассемблерной программы, секции кода и данных. Вычисление арифметических выражений:

пересылка данных, арифметические операции, регистр флагов.

5. Передача управления, условные и безусловные переходы.
6. Проблемы организации вызова функций. Аппаратная поддержка стека. Отображение вызова функции языка Си в язык ассемблера.
7. Восстановление алгоритмов и типов данных по бинарному коду программ. Декодирование и дизассемблирование.

II. ***Взаимосвязь языка Си, языка ассемблера и особенностей архитектуры IA32***

1. Отображение операторов разыменования указателя и взятия адреса из языка Си в язык ассемблера. Одномерные массивы, адресная арифметика. Размещение различных типов переменных языка Си в памяти компьютера.
2. Реализация арифметических операций над 64-разрядными целыми в архитектуре IA32. Знаковое и беззнаковое умножение в языке Си и языке ассемблера. Обоснование реализации умножения 64-разрядных знаковых целых чисел на 32-разрядных регистрах.
3. Условная передача данных: goto-форма представления ветвлений и циклов. Отображение операторов цикла языка Си в язык ассемблера. Организация цикла с помощью инструкции LOOP.
4. Различные способы отображения оператора switch в язык ассемблера: последовательность if-else, таблица переходов, дерево выбора.
5. Многомерные и многоуровневые массивы. Расположение в памяти, способы работы с отдельными элементами. Простейшие методы оптимизации, применяемые при работе

с массивами. Машинно-независимые и машинно-зависимые оптимизации.

6. Организация работы со структурами и объединениями языка Си на уровне языка ассемблера. Доступ к полям. Выравнивание данных.
7. Побитовые логические инструкции. Инструкции сдвига и вращения. Битовые поля структур.
8. Соглашение о вызове функций cdecl. Распределение памяти во фрейме функции. Организация рекурсии. Реализация различных классов памяти языка Си.
9. Возвращаемое значение в соглашении cdecl, возврат структур из функции в компиляторе gcc. Выравнивание фреймов в стеке. Организация вызова функций стандартной библиотеки языка Си из ассемблерного кода. Функции с переменным числом параметров.
10. Ускорение вызова функций и возврата из них: реализация вызова функций без указателя фрейма, соглашение fastcall.
11. Вызов функции по указателю. Обратный вызов. Ассемблерные вставки в Си-программах.
12. Формат инструкций архитектуры IA32. Префиксы: модификаторы размера операнда и адреса, префиксы повторения. Реализация операций над строками.
13. Представление чисел с плавающей точкой. Стандарт IEEE 754. Операции над числами с плавающей точкой. Округление чисел.
14. Сопроцессор FPU x87. Аппаратный стек регистров. Числа с плавающей точкой в языке Си: типы, доступ к результатам сравнения чисел, передача в качестве параметров функции, возвращаемое значение.

III. ***Архитектура компьютера***

1. Организация аппаратного обеспечения компьютера. Логические вентили. Закон Мура. Закон Гроша.
2. Устройство оперативной памяти. Статическая и динамическая память. Расслоение памяти.
3. Организация ввода/вывода. Шины, их характеристики, примеры шин: фронтальная шина, PCI, USB, SATA.
4. Периферийные устройства: НЖМД. Устройство, организация доступа к хранимой информации, емкость, временные характеристики доступа. Организация ввода/вывода через пространство портов и через память. Прямой доступ к памяти периферийных устройств. Твердотельные диски.
5. Тенденции в развитии запоминающих устройств. Локальность в программах. Иерархическая организация памяти, кэширование. Кэш-память процессора, способы ее организации: кэш прямого отображения, N-канальный ассоциативный кэш. Метрики производительности кэш-памяти.
6. Способы оценки производительности компьютеров и программ. Аппаратные средства измерения времени. Синтетический тест: оценка производительности памяти современного компьютера.
7. Конвейерная обработка команд. Приостановка и опустошение конвейера. Суперскалярная архитектура. CISC и RISC архитектуры.
8. Совместимость в программном и аппаратном обеспечении. Обратная совместимость процессоров. Включение/выключение линии A20.
9. Многозадачная работа компьютера: требования к аппаратуре, работа с памятью в защищенном режиме. Прерывания: программные, аппаратные, исключения.

10. Сегментная и страничная организация памяти. Логические, линейные и физические адреса памяти.
11. Режимы работы современного процессора архитектуры Intel64/AMD64: реальный, защищенный, «длинный», режим совместимости, режим системного управления. Процесс загрузки компьютера.

IV. *Система программирования языка Си и ее связь с архитектурой компьютера*

1. Компиляция и интерпретация. Система программирования языка Си. Программные инструменты, используемые при разработке Си-программ.
2. Схема работы ассемблера. Статическая компоновка программы. Схема работы компоновщика: разрешение символов и перемещение кода. Типы объектных файлов (модулей). Формат ELF.
3. Обработка компоновщиком нескольких символов с одинаковыми именами: сильные и слабые символы.
4. Организация многомодульной программы, роль заголовочных файлов. Автоматизация сборки многомодульной программы: утилита make.
5. Статические библиотеки. Компоновка со статическими библиотеками. Загрузка исполняемого файла в память.

V. *Рекомендованная литература*

1. Рэндал Э. Брайант, Дэвид О'Халларон. Компьютерные системы: архитектура и программирование (Computer Systems: A Programmer's Perspective). БХВ-Петербург, 2005 г. — 1186 стр.

2. Кип Р. Ирвин. Язык ассемблера для процессоров Intel. 4-е издание. Вильямс, 2005. — 912 стр.
3. Э. Таненбаум. Архитектура компьютера. 5-е изд. СПб.:Питер, 2007. — 844 стр.
4. Барри Брэй. Микропроцессоры Intel: 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium 4. Архитектура, программирование и интерфейсы. (The Intel Microprocessors 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium 4. Architecture, Programming,and Interfacing). БХВ-Петербург, 2005 г. —1328 стр.
5. Рудольф Марек. Ассемблер на примерах. Базовый курс. СПб: Наука и Техника, 2005. —231 стр.
6. Юров Виктор Иванович. Assembler. 2-е изд. Питер, 2010 — 640 стр.
7. Юров Виктор Иванович. Assembler: Практикум. 2-е изд. Питер, 2007 год. —400 стр.
8. Е.А. Кузьменкова, В.С. Махнычев, В.А. Падарян Семинары по курсу «Архитектура ЭВМ и язык ассемблера» (методическое пособие) Часть 1. — М. Издательский отдел факультета ВМК МГУ, 2012.

2. ВАРИАНТЫ ЭКЗАМЕНАЦИОННЫХ РАБОТ И КОЛЛОКВИУМОВ

2.1. ВАРИАНТ_1. 2012 г. (“Алгоритмы и алгоритмические языки”)

1. Составьте диаграмму машины Тьюринга (МТ), которая в непустом входном слове над алфавитом A_3={*0*, *1*, *2*} меняет местами символы в паре подряд стоящих символов (например, входное слово `02121` преобразуется в слово `20211`). Пары символов отсчитываются от левого края (первого символа) слова. Лента МТ бесконечна только справа. В начальном состоянии головка МТ обозревает пустую ячейку сразу после входного слова, в конечном состоянии – пустую ячейку сразу после выходного слова. Можно предполагать, что слева от входного слова на ленте есть одна пустая ячейка. При построении можно использовать элементарные машины *l, r, L, R, символ* (сдвиг головки на одну ячейку влево/вправо, сдвиг головки на одно слово влево/вправо, запись символа в ячейку, соответственно).

2. Дано описание:

```
struct list {int key; struct list *next;};
```

Напишите функцию

```
void move_elts_keys (struct list **src,
                     struct list **dst),
```

которая переносит из односвязного списка `src` в односвязный список `dst` все элементы с чётными ключами, удаляя их при этом из исходного списка. Перенесенные элементы должны добавляться в начало списка `dst` в произвольном порядке. Относительный порядок изначальных элементов обоих списков должен остаться неизменным. Считайте, что заголовочного элемента в обоих списках нет, списки могут быть пустыми.

3. Пусть `int x = 42; int y = -105;`

Для каждого из **6** отдельных независимых выражений указать ***его значение*** и ***побочные эффекты*** (если они есть) либо “ошибка”, если выражение ошибочно.

```
1) y %= x / 5
2) x ^ ~y & 65
3) y <<= (--x || --y)
4) (y = 1) && (y = 2)
5) x++ + --y
6) x += y, y = x - y, x -= y
```

4. Что будет напечатано при выполнении программы:

```
#include <stdio.h>
int x = 2, y = 1;
int a[3] = { 1, 2 };
int f (int x, int **p) {
  static int c = 2;
  if (c--)
    **p += x;
  else
    ++x;
  return (*p)++ - a - x;
}
int main (void) {
  int x = y, c = 0;  int *p;
  p = &a[x];
  for (int y = 2; y >= c; --y, ++x)
    c = f (y, &p);
  printf ("%d %d %d %d %d\n",
          a[0], a[1], a[2], x, y);
  return 0;
}
```

5. Перепишите приведенный фрагмент программы с использованием единственного оператора **switch** (дополнительные условные операторы использовать запрещено):

```
if (i == -1)
  j = i;
```

```
else if (3 >= i && i >= 1)
  j = 10 - i;
else
  abort ();
```

6. С использованием единственного цикла **for** напишите фрагмент программы, который для массива **int** a[N] формирует в переменной pairs количество пар соседних элементов, имеющих разную четность. Другие операторы циклов и операторы перехода использовать запрещено.

7. Напишите функцию

int scalar (**void**),

которая считывает со стандартного потока ввода непустую последовательность целых чисел, заканчивающуюся нулем (нуль не входит в последовательность, количество ненулевых элементов четно, элементы нумеруются с 1). Функция возвращает значение скалярного произведения двух векторов, составленных из элементов последовательности: первый вектор – из элементов с нечетными номерами, второй вектор – из элементов с четными номерами. В решении можно использовать не более одного оператора цикла. Запрещается использовать операторы перехода.

8. Напишите функцию

void erase (**char** *s, **const char** *w),

на вход которой дается строка ***s*** и слово ***w***. Слово – непустая строка из латинских букв. Строка ***s*** представляет собой последовательность слов, разделенных одним пробелом. Перед первым словом и после последнего пробелов нет. Функция должна удалить из строки ***s*** все слова, в которые слово ***w*** входит как подстрока. Относительный порядок всех остальных слов должен остаться неизменным. Измененная таким образом строка ***s*** должна удовлетворять всем тем же ограничениям, что и входная. Память под все временные массивы должна выделяться динамически и освобождаться в конце работы функции.

9. Дано описание:

struct avltree {**int** x; **struct** avltree *left, *right;};

Напишите функцию

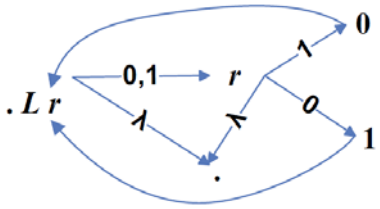
```
struct avltree *build (int h),
```

которая строит АВЛ-дерево заданной высоты ***h***, содержащее минимальное количество узлов ***n*** из всех АВЛ-деревьев с этой высотой. Значения ключей в элементах АВЛ-дерева должны быть от 1 до ***n***.

10. Нарисуйте дерево цифрового поиска для алфавита

{***В***, ***М***, ***И***, ***К***}, которое содержит следующие ключи: ***КИМ, МИКИ, ВИМ, ИМ, ИК, ВК, МИМ, ВМК, В, МИ, ВМИК, МИМИ.***

2.2. ВАРИАНТ_2. 2012 г. (“Алгоритмы и алгоритмические языки”)



1. Напишите нормальный алгоритм Маркова (не более **7** правил подстановки), эквивалентный заданной диаграмме Тьюринга над алфавитом A_2={*0*, *1*}. Диаграмма использует элементарные машины *l, r, L, R, символ* (сдвиг головки на одну ячейку влево/вправо, сдвиг головки на одно слово влево/вправо, запись *символа* в ячейку, соответственно). В начале работы головка МТ расположена сразу после входного слова.

2. Дано описание:

```
struct list {int key; struct list *next;};
```

Напишите функцию

```
struct list * merge_lists(struct list *l1,
                          struct list *l2),
```

которая объединяет два списка, вставляя элементы списка ***l2*** между элементами списка ***l1***: результирующий список состоит из первого элемента списка ***l1***, первого элемента списка ***l2***, второго элемента списка ***l1***, второго элемента списка ***l2*** и т.д. Если элементы одного из списков закончились, далее в результирующем списке должны идти элементы другого списка по порядку. Функция возвращает указатель на голову результирующего списка. Считайте, что заголовочного элемента в обоих

списках нет, списки могут быть как пустыми, так и непустыми. Дополнительную динамическую память выделять запрещается.

3. Пусть `int x = 11; int y = -80;`

Для каждого из **6** отдельных независимых выражений указать ***его значение*** и ***побочные эффекты*** (если они есть) либо “ошибка”, если выражение ошибочно.

```
1) y %= ++x
2) x | (-y ^ 1)
3) y <<= (x -= 10) && (x -= 11)
4) x++ + --x
5) x++ + --y
6) x++, --x, x + x, y + x
```

4. Что будет напечатано при выполнении программы:

```
#include <stdio.h>
int x = 5, y = 2;
int a[5] = { 1, 2, 3 };

int f (int *p, int *q) {
  static int a = 3;

  if (--a)
    return *p++ += *q--;
  return *q++ -= *p--;
}

int main (void) {
  int x = 2 * a[--y], b = 0;

  for (int y = x; y >= b; --y, ++x)
    b = f (a + b, a + y);
  printf ("%d %d %d %d %d\n",
          a[0], a[1], a[2], a[3], a[4]);
  printf ("%d %d\n", x, y);
  return 0;
}
```

5. С использованием оператора цикла **`for`** (возможно, нескольких) напишите фрагмент программы, который по заданному массиву

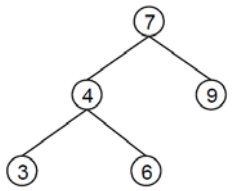
`int` `a[N]` формирует в переменной `sum` сумму $\sum_i x_i y_i$,

где x_i – i-й отрицательный элемент массива ***a***, считая от начала массива, а y_i – i-й положительный элемент массива ***a***, считая от конца массива. Количество положительных и отрицательных элементов в массиве ***a*** одинаково и больше нуля. Операторы перехода и другие операторы цикла использовать запрещено.

6. Дано описание:
`struct` `tree {`**`int`** `key;` **`struct`** `tree *left, *right;};` Не используя операторы цикла и перехода, глобальные и статические переменные, напишите функцию
`void` `large_sons (`**`struct`** `tree *t),`
которая печатает на стандартный поток вывода ключи тех элементов из двоичного дерева ***t***, ключи которых больше, чем ключи их родителей. Дерево ***t*** может быть как пусто, так и не пусто. Для корня дерева ничего печатать не нужно.

7. Напишите функцию
`int` `count (`**`const char`** `*s),`
которая подсчитывает все вхождения подстроки 2012 в строку ***s*** (например, для строки `"28012012-пересдача"` функция должна вернуть 1). Строка ***s*** может состоять из любых символов и быть как пустой, так и не пустой. Размер требуемой для функции памяти не должен зависеть от длины строки.

8. В соответствии с алгоритмом вставки и балансировки АВЛ-деревьев, вставьте в указанное АВЛ-дерево вершину с ключом **5**. Нарисуйте полученное АВЛ-дерево и укажите для каждой вершины этого дерева показатель сбалансированности.



2.3. Коллоквиум №2. 2012 г. (“Алгоритмы и алгоритмические языки”)

1. Пусть определены следующие переменные:

```
int x = 2, y = 4, z = 6;
short n = 0x7FFF; unsigned short m = 0x7FFFU;
struct S { struct P { int x, y; } p; struct S *n; } s
= { { -10, -20 }, &s };
unsigned int a[10] = { 2, 4, 6, 8, 10, 12, 14, 16,
18, 20 };
unsigned int *p, **pp;
```

Для каждого из **8** отдельных независимых выражений указать ***его значение*** и ***побочные эффекты*** (если они есть) либо “ошибка”, если выражение ошибочно.

```
1) x | y ^ z
2) n + 4
3) m <<= 1
4) s.p.x = x + s.n->p.y
5) *(a + 4 * x)
6) p = a + 1, **(pp = &p)
7) *pp = &a[2], *(*pp + y + a[2])
8) p = a + a[3], s.p.y = ++p - a
```

УКАЗАНИЕ.

В задачах 2-4 определите все необходимые переменные и типы. Считайте, что вызовы функций выделения памяти всегда заканчиваются успешно.

2. Напишите функцию, принимающую единственный параметр – массив строк (указателей типа **`char *`**). Последний элемент массива содержит нулевой указатель. Функция должна возвратить указатель на размещенный в динамической памяти массив структур, в которых первое поле есть указатель на копию одной из входных строк, а второе поле – количество раз, которое данная строка встречалась во входном массиве (с

учетом регистра символов). Одинаковые строки (с учетом регистра символов) должны содержаться в выходном массиве только в одном экземпляре, поле с указателем может указывать на любую из них. Выходной массив должен быть отсортирован сначала по убыванию по второму полю, а затем в лексикографическом порядке по первому полю.

3. Напишите функцию, которая удаляет из односвязного списка все элементы с нечетными номерами. Порядок следования всех остальных элементов должен остаться неизменным. Память, занятая удаляемыми элементами списка, должна быть освобождена с помощью вызова функции `free`. Считайте, что заголовочного элемента в списке нет и элементы списка нумеруются с 1. Ваша функция должна соответствовать прототипу

```
void remove_odd (struct list **);
```

где `struct list` описывает элемент списка с целочисленным ключом.

4. Напишите функцию, принимающую три параметра – массив целых чисел, его длину и указатель на целое число. Элементы массива образуют (возможно пустое) двоичное дерево (для элемента `a[i]` его детьми в двоичном дереве являются элементы `a[2*i+1]` и `a[2*i+2]`, элемент `a[0]` – корень дерева). Функция должна возвратить указатель на построенное в динамической памяти (в виде рекурсивной ссылочной структуры) двоичное дерево, в котором для каждого узла детьми являются те же числа, что и во входном массиве (или нулевой указатель для пустого дерева). По указателю, который является третьим параметром, необходимо записать 1, если построенное дерево является двоичным деревом поиска, и 0 в противном случае.

2.4. ВАРИАНТ_1. 2011 г. (“Алгоритмы и алгоритмические языки”)

1. а) Дать определение эквивалентности двух алгоритмов в некотором алфавите.

б) Ответить на вопрос: эквивалентны ли в алфавите $\{a, b\}$ указанные справа нормальные алгоритмы *T1* и *T2*?

в) Обосновать ответ на вопрос *б*).

$$T1: \begin{cases} a \to b \\ b \to a \end{cases}$$

$$T2: \begin{cases} a \to aa \\ b \to bb \end{cases}$$

2. Дано описание:

```
struct list { int key; struct list *next; } *A;
```

Не используя операторы цикла и перехода, а также глобальные и статические (static) переменные, описать функцию ***insert(L, x)***, которая вставляет в упорядоченный по возрастанию ключей список ***L***, новый элемент, имеющий в поле *key* ключ ***x*** (x – целое, не содержащееся ранее в списке). После вставки список остается упорядоченным по возрастанию. Список ***L*** – однонаправленный, некольцевой, без заглавного звена; звенья списка имеют тип `struct list`, все ключи – различны. Выписать также пример вызова функции для вставки в описанный выше список ***A*** ключа ***5***.

3. Пусть ***int x = 10 ; int y = 20;***
Для каждого из **6** отдельных выражений указать ***его значение*** и ***побочные эффекты*** (если они есть) либо “ошибка”, если выражение ошибочно.

1) `x +=(y = 1), y = 2`

2) `x ^= (x & y) + 4`

3) `x +=((y=1)||(y=2))`

4) `!x ? x : y`

5) `x %= y / 4`

6) `x += y++`

4. Что будет напечатано при выполнении программы:

```
#include <stdio.h>
int a[] = {1, 2, 3}; int b = 0, c = 2;
int p (int *a, int x) {
  static int c = 3;
  b = *++a; *a = b + x;
  return --c;
}
int main (void) {
  int i, b = c;
```

```
  for (i = 0; i < b; ++i)
    b = p (a + i, b);
  printf ("%d %d %d %d %d\n",
          a[0], a[1], a[2], b, c);
  return 0;
}
```

5. Дано описание:

```
struct tree { int x; struct tree *left, *right; };
```

Не используя операторы цикла и перехода, а также глобальные и статические (static) переменные, описать функцию

```
int is_avl (struct tree *T),
```

которая возвращает ненулевое значение, если заданное двоичное **дерево поиска** *T* является АВЛ-деревом, и ноль – в противном случае. Разрешается описывать дополнительные функции (с соблюдением тех же ограничений). Исходное дерево T портить запрещается.

6. Верно ли решена задача: «найти сумму первых 100 натуральных чисел»? Если неверно – объяснить, почему.

```
for (i = 0, sum = 0; i++, i < 100; sum += i);
```

7. а) Дайте определение применимости алгоритма к слову в алфавите.
б) Напишите нормальный алгоритм Маркова, в котором ***не более 4 формул*** подстановки и который применим к тем и только тем словам в алфавите $\{a,b,c\}$, к которым неприменим алгоритм, указанный слева.

$$\begin{cases} c \mapsto c \\ b \to b \\ a \to a \end{cases}$$

8. Верно ли утверждение: «действие оператора ***continue;*** в приведенных ниже примерах эквивалентно действию оператора ***goto next***; » (здесь ***E, E1, E2, E3*** - выражения допустимого в этом случае типа ; ***S*** - произвольный оператор

```
а) while (E) {S; for (E1; E2; E3)
                 {S; continue; S; next: ;} S;}
б) do { S; continue; S; } while (E); next: ;
в) for ( E1; E2; E3) { S; continue; S; next: ;}
```

9. Даны две перемешанные таблицы `T1[13]` и `T2[13]` с функцией первичного перемешивания `I=K%13`, функцией вторичного перемешивания `I=(I+4)%13` , где ***K*** – ключ. В таблицы ***T1*** и ***T2*** занесены следующие ключи:

Адрес	0	1	2	3	4	5	6	
T1		23				27	19	
T2		40				14		

	7	8	9	10	11	12
			18	6		
			5	36		

Дополнить таблицу ***T2*** ключами из таблицы ***T1***, причем, порядок записи ключей такой же, как и был при заполнении таблицы ***T1***. Указать содержимое таблицы ***T2*** после дополнения:

10. Ответить на вопрос: может ли в дереве Фибоначчи высоты 10 существовать вершина, левое поддерево которой имеет на 50 вершин больше, чем правое поддерево? (Считать, что пустое дерево имеет высоту 0.) Ответ обосновать.

2.5. ВАРИАНТ_1. 2012 г. (“Архитектура ЭВМ и язык ассемблера”)

1. Инструкции 1-6 выполняются последовательно. Выпишите в шестнадцатеричном формате значение регистра `EAX` после выполнения помеченных инструкций.

```
section .bss
   y resw 4
section .data
   x dd 0xFEE1DEAD

section .text
   movsx eax, word [x + 2]  ; 1
   imul  eax, eax, 16       ; 2 - (А)
   ror   eax, 24            ; 3
   mov    al, 42            ; 4 - (Б)
   mov   word [y + 5], ax   ; 5
   mov   eax, dword [y + 4] ; 6 - (В)
```

2. Для данного фрагмента ассемблерного кода восстановить соответствующий код на языке Си.

```
   mov   eax, dword [a]
   mov   eax, dword [eax]
   mov   ecx, dword [b]
   lea   eax, [eax + 4 * ecx]
   movsx ax, byte [eax]
   mov   word [c], ax
```

```
static ________________ a;
static ________________ b;
static ________________ c;

________________ = ________________;
```

3. Для данного фрагмента ассемблерного кода восстановите соответствующий код на языке Си, заполнив пропуски.

```
        mov     eax, dword [d]
        test    eax, eax
        jle     .L5
        xor     edx, edx
        jmp     .L4
.L3:
        add     edx, 1
        cmp     dword [d], edx
        jle     .L5
.L4:
        mov     eax, dword [b+edx*4]
        add     eax, dword [a+edx*4]
        test    eax, eax
        mov     dword [c+edx*4], eax
        jns     .L3
        neg     eax
        mov     dword [c+edx*4], eax
        add     edx, 1
        cmp     dword [d], edx
        jg      .L4
.L5:
```

```
#define N ...

static _________________ a[N];
static _________________ b[N];
static _________________ c[N];
static _________________ d;

   for (__________; __________; __________) {
      ____________________;
      if (_______________) {
         ___________________;
      }
  }
```

4. Си-программа, использующая структуру `order`, была скомпилирована на платформе `IA-32/Linux`.

```
typedef struct t_order {
   char status;
   int vendorId;
   char partNo[16];
   struct usrInfo * customer;
} order, *p_order;
```

Выпишите значения:

А) `sizeof(order)`
Б) Смещение поля `customer`
Реализуйте приведенный Си-код на языке ассемблера:

В)
```
static order o;
o.partNo[10] = 0;
```

Г)
```
static p_order po;
```

```
    static int x;
    x = po->vendorId;
```

5. Компилятор построил для тела Си-функции f следующий ассемблерный код. Исходя из этого кода и того, что было использовано соглашение `cdecl`, восстановите заголовок функции: типы параметров, их порядок, тип возвращаемого значения.

```
     mov     edx, dword [ebp+16]
     movzx   ecx, byte [ebp+20]
     mov     ebx, dword [ebp+12]
     mov     eax, edx
     sal     eax, cl
     mov     dword [ebx], eax
     mov     eax, dword [ebp+8]
     mov     dword [eax], edx
     mov     eax, dword [ebx]
```

```
___________ f(__________________ ,
              __________________ ,
              __________________ ,
              __________________ ) {
    *w = y << x;
    *v = y;
    return *w;
}
```

6. Реализуйте на языке ассемблера заданную функцию. Запрещено пользоваться командами ввода/вывода из файла `io.inc`. Перед вызовом функции `g` стек уже выровнен должным образом.

```
int g(short *pnum) {
   int tmp;
   printf("%d\n", tmp = scanf("%hd", pnum));
   return tmp;
}
```

7. Для данного фрагмента Си-кода приведите соответствующий ассемблерный код, учитывая побочные эффекты. Значение выражения поместите в регистр `EAX`.

```
  static int a, b, c;
```

```
++a && (c = b /a );
```

8. Перепишите фрагмент кода без использования строковых инструкций, но с сохранением итогового значения регистра ECX и флага ZF.

```
cld
mov esi, dword [a]
mov edi, dword [b]
mov ecx, 16
repne cmpsw
```

9. Используется 8-и битный формат, удовлетворяющий требованиям стандарта IEEE 754: знаковый бит, 3 бита – порядок, 4 бита - мантисса. Заполните таблицу. Округление выполнять к ближайшему четному.

Число	*Битовое представление*
Наибольшее нормализованное	
Наименьшее денормализованное	
2/5	
13	

10. Память модельного компьютера состоит из 32 однобайтовых ячеек. 2-канальный ассоциативный кэш имеет следующие характеристики: B=2 байта в блоке, S=2 канала, E=2 блока в канале, вытесняется блок с наиболее давним использованием (LRU). В начальный момент времени кэш пуст. Определите, что происходит при последовательном обращении к следующим адресам памяти (промах/попадание).

Адрес	Промах/Попадание
2 = 00010b	
3 = 00011b	
8 = 01000b	
16 = 10000b	
29 = 11101b	
8 = 01000b	

28 = 11100b	

2.6. Коллоквиум №1. 2012 г. (“Архитектура ЭВМ и язык ассемблера”)

1. Выпишите значение регистра AL в шестнадцатеричной системе и в виде десятичного числа (знакового и беззнакового), а также значения флагов CF, OF, ZF, SF после выполнения следующих инструкций.

```
MOV AL, 145
ADD AL, 157
```

2. Выписать значение регистра EAX после выполнения следующего кода.

```
section .data
   a dw 0x0DEC
   b dw 0x4A6F
   c dw 0x7921
   d dw 0xFEFF
```

```
section .text
   mov   ebx, dword [c]
   mov   eax, -1
   movzx ax,  bh
```

3. Для данного фрагмента ассемблерного кода восстановить соответствующий код на языке Си

```
mov     ecx, dword [a]
movsx   edx, word [d]
mov     eax, ecx
imul    eax, dword [b]
sub     eax, edx
movsx   ebx, word [c]
cdq
idiv    ebx
sub     ecx, eax
mov     dword [a], ecx
```

```
static _________________ a;
static _________________ b;
static _________________ c;
static _________________ d;

______ =
_____________________________;
```

4. Для данного фрагмента ассемблерного кода восстановить соответствующий код на языке Си.

mov edx, dword [b] mov eax, dword [a] mov edx, dword [edx] lea eax, [4*eax] neg eax mov eax, dword [edx+eax] mov dword [c], eax	static _____ a; static _____ b; static _____ c; _____ = _____;

5. Для данного фрагмента ассемблерного кода восстановить соответствующий код на языке Си

```
   xor     ecx, ecx
   xor     eax, eax
.L3:
   mov     edx, dword [a+ecx*4]
   cmp     edx, 1024
   jle     .L2
   mov     dword [a+ecx*4], 1024
   mov     edx, 1024
.L2:
   add     ecx, 1
   add     eax, edx
   cmp     ecx, 16
   jne     .L3
   mov     dword [sum], eax
```

```
#define N ____
#define C ____

   static int a[N];
   static int sum;

   _____;
   for (_____; _____; _____) {
      if (_____) {
         _____;
      }
      _____;
   }
```

6. Для данного фрагмента Си-кода приведите соответствующий ассемблерный код. Необходимо учесть побочные эффекты. Значение выражения поместите в регистр EAX.

```
static int a, b, c;

(a = ++b) && (c *= 7);
```

2.7. Коллоквиум №2. 2012 г. (“Архитектура ЭВМ и язык ассемблера”)

1. В приведенной ниже Си-программе (скомпилирована на платформе IA-32/Windows), использующей структуру данных `gadget`, выпишите значения: :

А) `sizeof(struct gadget)`
Б) Смещение поля `prototype`
В) `sizeof(union i_face)`
Г) Смещение поля `magic`

```
struct gadget {
   int id;
   short weight;
   union i_face {
      double resistor;
      char magic;
      int* i_desc;
      long pin_map[2];
   } connector;
   struct gadget * prototype;
};
```

2. Компилятор построил для тела Си-функции **f** следующий ассемблерный код.
Исходя из этого кода, и того, что было использовано соглашение **cdecl,** восстановите заголовок функции: типы параметров, их порядок, тип возвращаемого значения.

```
mov     ecx, dword [ebp+12]
mov     eax, dword [ebp+20]
movsx   edx, byte  [ebp+8]
sal     edx, 2
mov     ebx, ecx
sub     ebx, edx
mov     dword [eax], ebx
movsx   edx, word [ebp+16]
mov     dword [ecx], edx
```

```
___________ f(___________________ ,
              ___________________ ,
              ___________________ ,
              ___________________ ) {
   *c = d - a;
   *d = b;
   return c;
}
```

3. Реализуйте на языке ассемблера заданную функцию.

```
__attribute__((fastcall)) int f(int x, int y) {
   int t = (x / y) >> 10;
   return t;
}
```

4. Реализуйте на языке ассемблера заданную функцию. Перед вызовом функции g стек уже выровнен должным образом.

```
long g(long *p) {
   long tmp;
   scanf("%d %d", &tmp, p);
   return *p – tmp;
}
```

5. Выписать значение регистров ESI, EDI и ECX после выполнения следующего кода.
Метке **team** соответствует адрес **0x4096**.

```
section .data
BUFSIZE equ 16
   team: db “Charlie Brown“
   times   BUFSIZE-$+team db 0
         db “Snoopy“
   times 2*BUFSIZE-$+team db 0
         db “Linus“
   times 3*BUFSIZE-$+team db 0
         db “Lucy“
   times 4*BUFSIZE-$+team db 0


section .text
  cld
  mov esi, team
  mov edi, team
  add esi, 3*BUFSIZE
  add edi, 2*BUFSIZE
  mov ecx, 16
  repe cmpsb
```

6. Используется 10-ти битный формат, удовлетворяющий требованиям стандарта **IEEE 754**: знаковый бит, порядок - 4 бита, мантисса - 5 битов. Заполните пустые ячейки таблицы.

Описание	Точное значение	Битовое представление
Ноль	0.0	0_0000_00000
Наибольшее нормализованное		
Наименьшее положительное		
1/3		
Номер вашей группы [101 – 106]		

2.8. ВАРИАНТ_1. 2011 г. (“Архитектура ЭВМ и язык ассемблера”)

1. Выпишите значение регистра AL в шестнадцатеричной системе и в десятичной системе, рассматривая содержимое регистра как беззнаковое число и как знаковое число, а также значения флагов CF, SF, OF и ZF после выполнения следующих инструкций.

а)
```
mov al, -20
add al, 144
```
б)
```
mov al, 80
sub al, 86
```

2. Выписать в шестнадцатеричном виде значение регистра EAX, после выполнения следующей инструкции.

```
section .data
   a dw 0xDEAD
   b dw 0xF00D
   c dw 0xCAFE
   d dw 0xBABE
```

```
section .text
   movsx eax, word [a + 1]
```

3. Для приведенного фрагмента Си-кода написать соответствующий фрагмент ассемблерной программы.

```
static int *p[10];
static int x;
x = *p[8] + 1;
```

4. Восстановите по приведенному ниже фрагменту ассемблерной реализации соответствующий фрагмент Си-кода. Необходимо правильно заполнить пробелы, указав типы переменных, управляющие выражения операторов if и вычисляющихся в них выражений.

```
mov  ebx, dword [a]
mov  ecx, dword [b]
cmp  ebx, ecx
```

```
        jg   .L7
        jl   .L8
        xor  eax, eax
        jmp  .L9
.L8:
        mov  edx, dword [c]
        mov  eax, edx
        sar  edx, 31
        sub  ecx, ebx
        idiv ecx
        jmp  .L9
.L7:
        mov  edx, dword [c]
        mov  eax, edx
        sar  edx, 31
        sub  ebx, ecx
        idiv ebx
.L9:
        mov  dword [d], eax

...
```

```
static __________________ a;
static __________________ b;
static __________________ c;
static __________________ d;
if (__________) {
   d = _________________;
} else if (__________) {
   d = _________________;
} else {
   d = _________________;
}
```

```
...
```

5. Напишите ассемблерную программу, проверяющую симметричность статического массива (N элементов типа int, N – константа, N < 220): одинаковые значения имеют первый и последний, второй и предпоследний, и т.д. Выполнять ввод значений элементов массива и выравнивать стек не требуется. Если массив симметричный, на выходе из функции CMAIN регистр EAX должен иметь значение 0, в противном случае – 1. Запрещается использовать строковые инструкции и использовать стек. Программа должна содержать **не более 16 инструкций.** Учитываются все инструкции, включая ret.

6. В приведенном фрагменте Си-кода используются константы **H** и **J.** Исходя из построенного компилятором кода, восстановите значения констант **H** и **J.**

```
int array1[H][J];
int array2[J][H];

void copy_array(int x, int y) {
   array2[y][x] = array1[x][y];
}
```

```
copy_array:
   push ebp
   mov  ebp, esp
   mov  eax, dword [ebp + 8]
   mov  edx, dword [ebp + 12]
   lea  ecx, [eax + 4 * edx]
   lea  eax, [edx + 8 * eax]
   mov  eax, dword [array1 + 4 * eax]
   mov  dword [array2 + 4 * ecx], eax
   pop  ebp
   ret
```

7. Тело Си-функции **f**, использующей соглашение **cdecl,** было переведено компилятором в следующий код. Восстановите объявление функции **f**.

```
*b = x;
return c - y;
```

```
movsx ecx, byte [ebp + 8]
mov   eax, dword [ebp + 16]
mov   ebx, dword [ebp + 20]
mov   edx, dword [ebp + 12]
sub   eax, ecx
mov   word [edx], bx
```

8. Используя соглашение **cdecl,** напишите на языке Ассемблера реализацию Си-функции **seq**. Следует считать, что при вызове функции **seq** стек уже выровнен по 16-байтной границе.

```
int *seq(int size) {
   int *p = (int *) malloc(size * sizeof(int));
   for (int i = 0; i < size; i++) {
      p[i] = i;
   }
   return p;
}
```

9. а) нарисуйте схему логических вентилей, выполняющую сравнение двух битов

б) Оцените среднее время доступа для диска со следующими характеристиками

- **Скорость вращения 5 400 RPM**
- **Среднее время поиска дорожки 12 мс.**
- **Среднее количество секторов на дорожке 300.**

10. а) Укажите верную последовательность вызова драйвером компилятора gcc служебных действий.

Компоновщик, препроцессор, компилятор, ассемблер.
Препроцессор, компилятор, ассемблер, компоновщик.
Компилятор, ассемблер, препроцессор, компоновщик.
Компоновщик, компилятор, ассемблер, препроцессор.
Препроцессор, компилятор, компоновщик, ассемблер.

б) В сборке Си-программы участвуют два приведенных ниже файла: f1.c и f2.c. Отметьте верные высказывания.

Ошибка компоновки: два сильных символа f.
Ошибка компоновки: два сильных символа y.
Запись в y (файл f2.c) меняет значение переменных y и z в файле f1.c.
Запись в y (файл f2.c) меняет значение переменной y и не меняет z в файле f1.c.

Для переменных y будут отведены два различных места в памяти.
Запись в x (файл f1.c) необязательно будет менять значение переменной x в файле f2.c.

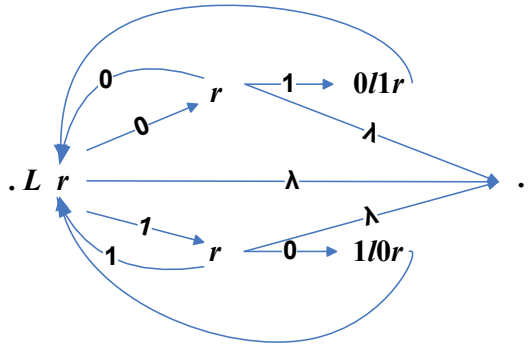
f1.c	f2.c

int x;
int y = 7;
int z;
 	
void f() {
…
}
	float x;
	double y;
	
	int f() {
	…
	}

3. ОТВЕТЫ, УКАЗАНИЯ И РЕШЕНИЯ

3.1. ВАРИАНТ_1. 2012 г. (“Алгоритмы и алгоритмические языки”)

1. Диаграмма выглядит следующим образом:



2. Решение:

```
void move_elts_keys (struct list **src,
                     struct list **dst) {
  struct list *p, *q;   /* p – предыдущий элемент,
                           q – текущий */
  p = NULL; q = *src;
  while (q) {
    if (q->key % 2 == 0) {
      struct list *t = q->next;  /* t – следующий
                                    из исходного списка */
      q->next = *dst;
      *dst = q;      /* Добавляем q в голову dst */
      if (p)
        p->next = t;  /* Если был предыдущий,
                         исправляем исходный список */
      else
```

```
        *src = t;      /* Иначе меняем его голову */
      q = t;           /* Следующий на обработку */
    } else {
      p = q;
      q = q->next;    /* Двигаем ссылки на текущий
                         и предыдущий элементы */
    }
  }
}
```

3. Ответы:

1) -1, y = -1

2) 106

3) -210, x = 41, y = -210

4) 1, y = 2

5) -64, x = 43, y = -106

6) -105, x = -105, y=42

4. Ответ: 1 4 1 3 1

5. Решение:

```
switch (i) {
  case -1: j = i; break;
  case 1: case 2: case 3: j = 10 - i;
  default: abort ();
}
```

6. Решение:

```
for (i = 0, pairs = 0; i < N - 1; i++)
  pairs += (a[i] % 2 != a[i+1] % 2 ? 1 : 0);
```

7. Решение:

```
int scalar (void) {
  int x, y, sum = 0;
  scanf ("%d", &x);
```

```
  while (x) {
    scanf ("%d", &y);
    sum += (x*y);
    scanf ("%d", &x);
  }
  return sum;
}
```

8. Решение:

```
void erase (char *s, const char *w) {
  char *olds = s;
  while (*s) {
    char *p = s; char c;
    while (isalpha (*p))
      p++;
    /* Теперь между s и p первое слово, p указывает
    на пробел или 0-терминатор */
    c = *p; *p = '\0'; /* чтобы можно было
                          использовать strstr */
    if (strstr (s, w)) {
      /* удаляем слово сдвигом конца строки в
      начало, также можно использовать memmove */
      if (c == '\0') {
        /* удаляемое слово было последним, надо
        записать 0-терминатор или сразу в s,
        или стереть предыдущий пробел, если текущее
        слово не первое */
        if (s > olds)
          s--;
        *s = 0;
      }
      else {
        char *q = s;
        *p = c; /* вернем стертый символ, это будет
                   пробел */
        p++; /* слово было не последнее, значит,
```

```
            это можно делать */
        do {
          *q++ = *p++;
        } while (*(p - 1));   /* Копируем и нулевой
                                 символ тоже */
      }
    } else {
      *p = c; /* вернем стертый символ, это будет
              пробел или 0-терминатор */
      s = *p ? p + 1 : p; /* если пробел, то
                           пропустим его */
    }
  }
}
```

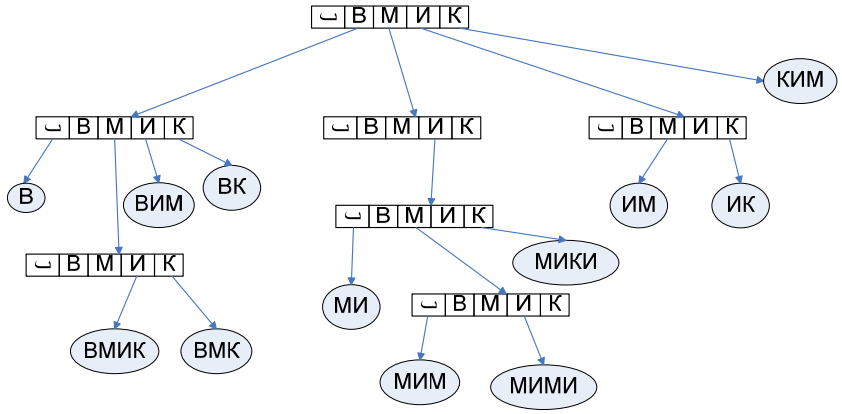
9. Решение: необходимо построить дерево Фибоначчи заданной высоты.

```
static struct avltree *build_rec (int h, int min, int
*p) {
  struct avltree *t = (struct avltree *) malloc
                  (sizeof (struct avltree));
  int last;
  if (h == 1) { /* листовое дерево */
    t->left = t->right = NULL;
    t->x = min;
    *p = min + 1;
    return t;
  }
  t->left = build_rec (h - 1, min, &last);
  t->x = last++; /* Левый сын занял от 1 до
                 last-1 элементов, корень - last */
  if (h == 2)
    t->right = NULL;  /* Правый сын нулевой */
  else
    t->right = build_rec (h - 2, last, &last);
  *p = last;
  return t;
```

```
}

struct avltree *build (int h) {
  int last;
  return build_rec (h, 1, &last);
}
```

10. Результирующее дерево:



3.2. ВАРИАНТ_2. 2012 г. (“Алгоритмы и алгоритмические языки”)

1. Заданная диаграмма описывает машину Тьюринга, которая инвертирует каждый второй символ из пары символов, пары отсчитываются, начиная от первого символа слова. Предлагаемый алгоритм Маркова:

```
*1→1#
*0→0#
#0→1*
#1→0*
# ↦
* ↦
  → *
```

2. Приведем одно из решений (возможно также значительно более короткое решение):

```
struct list * merge_lists (struct list *l1, struct
list *l2) {
```

```
  /* явно держим указатель на выходной
     список и вставляем по очереди */
  struct list *p = NULL, *h = NULL;
  int flag = 1;
  while (l1 || l2) {
    struct list *q = flag ? l1 : l2;
    if (p)
      p->next = q, p = p->next;
    else
      p = h = q;
    /* обязательно в таком порядке */
    if (flag)
      l1 = l1->next;
    else
      l2 = l2->next;
    p->next = NULL;
    flag = 1 - flag;
  }
  /* не забываем про конец списка */
  if (l1 || l2) {
    if (p)
      p->next = l1 ? l1 : l2;
    else
      h = l1 ? l1 : l2;
  }
  return h;
}
```

3. Ответы:

1) -8, x= 12, y = -8

2) 91

3) -160, x = -10, y = -160

4) ошибка

5) -70, x = 12, y = -81

6) -69, x= 11

4. Ответ: 3 2 0 0 0
8 1

5. Решение:

```
int sum = 0, ineg = 0, ipos = N - 1, i;
/* внешний цикл может быть бесконечным */
for (i = 0; i < N/2; i++) {
  for (; ineg < N && a[ineg] >= 0; ineg++)
    ;
  for (; ipos >= 0 && a[ipos] <= 0; ipos--)
    ;
  if (ineg == N || ipos == -1)
    break;
  sum += a[ineg++]*a[ipos--];
}
```

6. Решение:

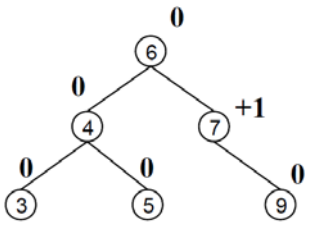
```
void large_sons (struct tree *t) {
  if (!t)
    return;
  if (t->left && t->left->key > t->key)
    printf ("%d\n", t->left->key);
  if (t->right && t->right->key > t->key)
    printf ("%d\n", t->right->key);
  large_sons (t->left);
  large_sons (t->right);
```

```
}
```

7. Решение:

```
int count (const char *s) {
  int res = 0, len = strlen (s) - 3, i;
  for (i = 0; i < len; i++)
    if (s[i] == '2' && s[i+1] == '0'
        && s[i+2] == '1' && s[i+3] == '2')
      res++;
  return res;
}
```

8. Результирующее дерево показано на рисунке.



3.3. Коллоквиум №2. 2012 г. (“Алгоритмы и алгоритмические языки”)

1. Ответы:

1) 2

2) 32771 или 0x8003

3) 65534 или 0xFFFE, m = 65534

4) -18, s.p.x = -18

5) 18

6) 4, p = &a[1], pp = &p

7) ошибка – *pp + y + a[2] = &a[12], разыменовывать этот указатель нельзя, т.к. в массиве a 10 элементов

8) 20, s.p.y = 20, p = &a[9]

2. Решение:

```
struct str {
  char *s;
  int cnt;
};

struct str *task2 (char ** a) {
  int i, j, n = 0, sz = 0;
  struct str *out = NULL;
  /* Строим массив структур */
  for (; *a; a++) {
    for (i = 0; i < n; i++)
      if (! strcmp (*a, out[i].s)) {
        out[i].cnt++;
        break;
      }
    if (i >= n) {
      /* Встретили новую строку */
      if (i >= sz) {
        out = realloc (out, 2 * sz + 1);
        sz = 2 * sz + 1;
      }
      out[n].s = strdup (*a);
      out[n++].cnt = 1;
    }
  }
  /* Сортируем массив */
  for (i = 0; i < n - 1; i++)
    for (j = i + 1; j < n; j++)
      if (out[i].cnt < out[j].cnt
          || strcmp (out[i].s, out[j].s) > 0) {
        struct str tmp = out[i];
```

```
        out[i] = out[j];
        out[j] = tmp;
      }
  return out;
}
```

3. Решение:

```
struct list {
  int key;
  struct list *next;
};

void remove_odd (struct list **pl) {
  int num = 1;
  struct list *p = *pl, *q = NULL;
  while (p) {
    if (num % 2 == 1) {
      struct list *t = p;
      if (!q)
        /* Удаляем первый элемент*/
        *pl = p->next;
      else
        /* Сдвигаем ссылку с предыдущего */
        q->next = p->next;
      p = p->next;
      free (t);
    } else {
      q = p;
      p = p->next;
    }
    num++;
  }
}
```

4. Решение:

```
struct tree {
  int key;
```

```
  struct tree *left, *right;
};

struct tree *build_tree (int *a, int i, int n) {
  struct tree *t;
  if (i >= n)
    return NULL;
  t = (struct tree *) malloc
                        (sizeof (struct tree));
  t->key = a[i];
  t->left = build_tree (a, 2*i + 1, n);
  t->right = build_tree (a, 2*i + 2, n);
  return t;
}
/* Функция проверяет, все ли вершины поддерева T
входят в необходимый диапазон значений */
int is_bin (struct tree *t, int *pmin, int *pmax) {
  int ans = 1;
  if (!t)
    return ans;
  /* Нулевое значение указателя означает отсутствие
     соответствующего ограничения */
  if ((pmin && t->key <= *pmin)
       || (pmax && t->key >= *pmax))
    return 0;
  ans = is_bin (t->left, pmin, &t->key);
  ans = ans && is_bin (t->right, &t->key, pmax);
  return ans;
}

struct tree *task4 (int *a, int n, int *pbin) {
  struct tree * t = build_tree (a, 0, n);
  *pbin = is_bin (t, NULL, NULL);
  return t;
}
```

3.4. ВАРИАНТ_1. 2011 г. (“Алгоритмы и алгоритмические языки”)

1. а) ***Определение*:** два алгоритма эквивалентны в алфавите A (на множестве слов из алфавита A), если:

- на одинаковых словах из A^* они одновременно либо останавливаются, либо зацикливаются (т.е. области их применимости совпадают); и для любого слова из области применимости их результаты совпадают.

Другие варианты ответа а1) . Возможно и такое определение эквивалентности - Два алгоритма, заданные в алфавите А, эквивалентны, если их области применимости совпадают, и для любого слова из области применимости эти алгоритмы получают одинаковый результат.

а2). Два алгоритма, заданные в алфавите А, эквивалентны, если они реализуют одно и то же преобразование (отображение) на множестве слов A^* ($A^* \to A^*$)

1. б) Ответ: **ДА.**

1. в) ***Обоснование:*** алгоритмы *T1* и *T2* применимы только к пустому слову, которое они и выдают в качестве ответа, а на любом непустом слове из алфавита {*a*, *b*} они зацикливаются (правда, зацикливаются по-разному, но это не играет никакой роли).

2. Возможны различные решения задачи (но использование рекурсии обязательно), приведем два варианта решения :

Вариант 2.1:

```
void
insert(struct list **L, int x) {
  struct list *p;
  if (*L == NULL || (*L)->key > x) {
// вставка в начало
    p = malloc(sizeof(*p));
    p->key = x;
    p->next = *L;
    *L = p;
  } else {
    insert(&((*L)->next), x);
```

```
  }
}
//   Вызов
insert(&A, 5);
```

Вариант 2.2

```
struct list *
insert(struct list *L, int x) {
  struct list *p;
  if (L == NULL || L->key > x) {
// вставка в начало
    p = malloc(sizeof(*p));
    p->key = x;
    p->next = L;
    L = p;
  } else {
    L->next = insert(L->next, x);
  }
  return L;
}
```

```
                    //   Вызов:
                A = insert(A, 5);
```

3. Ответы

1) 2, x :=11, y :=2

2) 14 , x :=14

3) 11, x :=11, y :=1

4) 20

5) 0, x :=0

6) 30, x :=30

4. Ответ: 1 4 5 1 2

5. *Решение:*

Пояснение. В непустом АВЛ-дереве у ***каждой*** вершины разность высот её левого и правого поддеревьев должна равняться 0 или ±1 (как следствие, в АВЛ-дереве любое поддерево – АВЛ-дерево).

Для каждой вершины надо проверять две вещи:
1) являются ли её левое и правое поддеревья АВЛ-деревьями;
2) модуль разности высот этих поддеревьев ≤ 1.)

Вариант решения

```
int
height(struct tree *T) {
  int hleft, hright;
  if (T == NULL) return 0;
  hleft = height(T->left);
  hright = height(T->right);
  return 1 + ((hleft > hright) ? hleft : hright);
}

int abs(int x) {
  return (x >= 0)? x : -x;
}
int is_avl(struct tree *T) {
  if (T == NULL) return 1;
  if (abs(height(T->left) - height(T->right)) > 1)
return 0;
  return is_avl(T->left) && is_avl(T->right);
}
```

6. Ответ: Неверно (или нет).
Объяснение: – в сумме будет отсутствовать число 100

7. Заданный : алгоритм неприменим к словам, где нет ***c,*** но и при этом есть ***a*** или ***b***.
а): Алгоритм применим к заданному слову в алфавите, если он остановится (не зациклится), начав работу над этим словом.

б): $\begin{cases} c \to c \\ b \mapsto b \\ a \mapsto a \\ \to \end{cases}$

8. Ответы :
```
а) Верно
б) Неверно (нет)
в) Верно
```

9. Порядок занесения ключей в таблицу T1 такой: 19, 6, 23, 27, 18**.**
Ответ:

Адрес	0	1	2	3	4	5	6	
T1		23				27	19	
T2	6	40			23	14	19	

	7	8	9	10	11	12
			18	6		
		27	5	36		18

10. Ответ: нет
Обоснование:
Число вершин в дереве Фибоначчи высоты h равно nf(h) = F_{h+2} -1 (где F_{h+2} - число Фибоначчи с номером h+2). Дерево Фибоначчи является АВЛ-деревом с минимальным числом вершин при заданной высоте h.
1) Любое поддерево в дереве Фибоначчи (д.Ф.) является д.Ф.
2) Для любого д.Ф. высоты h верна формула $N_0=0$, $N_1=1$, $N_h=N_{h-1}+N_{h-2}+1$, где N_h, N_{h-1} и N_{h-2} – число вершин соответственно в самом этом дереве и в его поддеревьях.
3) Поэтому заданный вопрос эквивалентен вопросу о том, есть ли такое $h\le10$, для которого выполняется условие $N_{h-1}–N_{h-2}>50$.
4) Поскольку $N_0=0$, $N_1=1$, $N_2=2$, $N_3=4$, $N_4=7$, $N_5=12$, $N_6=20$, $N_7=33$, $N_8=54$, $N_9=88$, $N_{10}=143$, …
то наибольшее значение разности $N_{h-1}–N_{h-2}$ при $h\le10$ равно $N_9–N_8=88–54=34<50$, поэтому такого значения h нет

3.5. ВАРИАНТ_1. 2012 г. (“Архитектура ЭВМ и язык ассемблера”)

1. Ответы:

А) **0xffffee10**

Б) **0xffee102a**

В) **0x00102a00**

2. Решение:

```
static signed char **a;
static int b;
```

```
static short c;
c = (*a)[4 * b];
```
Альтернативный вариант решения, не использующий индексное выражение:
```
c = *((*a) + 4 * b);
```

3. Решение:
```
#define N ...


static int a[N];
static int b[N];
static int c[N];
static int d;


   for (int i = 0; i < d; i++) {
      c[i] = a[i] + b[i];
      if (c[i] < 0) {
         c[i] = -c[i];
      }
  }
```

4. Ответы:
А) **28**
Б) **24**
Решение:
В)
```
mov byte [o + 18], 0
```
Г)
```
mov eax, dword [po]
mov eax, dword [eax + 4]
mov dword [x], eax
```

5. Решение:
```
   int f(int*            v,
```

```
        int*            w,
        int             y,
        unsigned char   x) {
    *w = y << x;
    *v = y;
    return *w;
}
```

6. Решение:

```
section .rodata
   fmt1 db “%hd”, 0
   fmt2 db `%d\n`, 0

section .text
g: push ebp
   mov epb, esp
   sub esp, 24
   mov [esp], fmt1
   mov eax, [ebp+8]
   mov [esp+4], eax
   call scanf
   mov [ebp-4], eax
   mov [esp+4], eax
   mov [esp], fmt2
   call printf
   mov eax, [ebp-4]
   mov esp, ebp
   pop ebp
   ret
```

7. Решение:

```
   inc dword [a]
   mov eax, dword [a]
   je .1
   mov eax, dword [b]
   cdq
   idiv dword [a]
   mov dword [c], eax
   test eax, eax
   cmovnz eax, 1
.1:
```

8. Решение:

```
   mov esi, dword [a]
   mov edi, dword [b]
   mov ecx, 16
.1:mov ax, word [esi]
   mov dx, word [edi]
   add esi, 2
   add edi, 2
   dec ecx
   cmp ax, dx
   jecxz .2
   jne .1
.2
```

Альтернативный вариант решения, использующий инструкцию loopne.

```
   mov esi, [a]
   mov edi, [b]
   mov ecx, 16
.1:mov ax, [esi]
   cmp ax, [edi]
   lea esi, [esi+2]
   lea edi, [edi+2]
   loopne .1
```

9. Ответ:

Число	*Битовое представление*
Наибольшее нормализованное	**0_110_1111**
Наименьшее денормализованное	**1_000_1111**
2/5	**0_001_1010**
13	**0_110_1010**

10. Ответ:

Адрес	*Промах/Попадание*
2 = 00010b	промах
3 = 00011b	попадание
8 = 01000b	промах
16 = 10000b	промах

29 = 111<u>0</u>1b	промах
8 = 010<u>0</u>0b	промах
28 = 111<u>0</u>0b	попадание

3.6. Коллоквиум №1. 2012 г. (“Архитектура ЭВМ и язык ассемблера”)

1. Ответ: AL = 0x2E (16-ричн.), 46 (10-тичн. зн.), 46 (10-тичн. б./зн.) CF = 1, OF = 1, ZF = 0, SF = 0

2. Ответ: EAX = 0xFFFF0079

3. Решение:
```
static int a;
static int b;
static short c;
static short d;

a = a – (a*b-d)/c;
```

4. Решение:
```
static int  a;
static int **b;
static int c;

c = *((*b) - a);
```

Для вычисляемого выражения допустима и такая форма записи:
```
c = (*b)[-a];
```

5. Решение:
```
#define N   16
#define C 1024

   static int a[N];
   static int sum;

   sum = 0;
   for (int i = 0; i != N; i++) {
```

```
      if (a[i] > C) {
         a[i] = C;
      }
      sum += a[i];
   }
```

6. Решение:

```
   INC DWORD [B]
   MOV EAX, DWORD[B]
   MOV DWORD [A], EAX
   TEST EAX, EAX
   JZ .END
   MOV EAX, DWORD [C]
   IMUL EAX, EAX, 7
   MOV DWORD [C], EAX
   TEST EAX, EAX
   JZ .END
   MOV EAX, 1
.END:
```

3.7. Коллоквиум №2. 2012 г. (“Архитектура ЭВМ и язык ассемблера”)

1. Поскольку в условии указана ОС Windows и в `union i_face` присутствует поле типа double, выравнивание для `union i_face` и для всей структуры `gadget` будет выполняться по границе в 8 байт.

Ответ:

А) sizeof(struct gadget) = 24
Б) Смещение поля prototype = 16
В) sizeof(union i_face) = 8
Г) Смещение поля magic = 8

2. Ответ:

```
int** f(signed char a,
                int* d ,
               short b ,
               int** c) {
   *c = d - a;
   *d = b;
   return c;
```

```
}
```

3. Решение:

```
f:     MOV EAX, ECX ; первый параметр
       MOV ECX, EDX ; второй параметр
       CDQ
       IDIV ECX
       SHR EAX, 10
       RET
```

4. Решение:

```
section .rodata
g.fstr db '%d %d', 0

section .text
CEXTERN scanf
g:     PUSH EBP
       MOV EBP, ESP
       SUB ESP, 24        ; общий размер фрейма 32 байта
       MOV DWORD [ESP], g.fstr        ; форматная строка
       LEA EAX, DWORD [EBP-4]                      ; tmp
       MOV DWORD [ESP+4], EAX
       MOV EAX, DWORD [EBP+8]                        ; p
       MOV DWORD [ESP+8], EAX
       CALL scanf
       MOV EAX, DWORD [EBP+8]
       MOV EAX, DWORD [EAX]
       SUB EAX, DWORD [EBP-4]
       MOV ESP, EBP
       POP EBP
       RET
```

5. Ответ:

А) ESI = 0x40C8
Б) EDI = 0x40B8
В) ECX = 14

6. Ответ:

Описание	Точное значение	Битовое представление
Ноль	0.0	0_0000_00000
Наибольшее	(63/32)*(2^7)=252	0_1110_11111

нормализованное		
Наименьшее положительное	(1/32)*(1/64)= 2^(-11)	0_0000_00001
1/3	21*2^(-6)	0_0101_01010

В рамках данной кодировки нечетные номера групп не могут быть точно представлены. Ниже приведен ответ, где для таких номеров выполнено округление к наименьшему представимому числу.

Группа №101	100.0	0_1101_10010
Группы №№102 и 103	102.0	0_1101_10011
Группы №№104 и 105	104.0	0_1101_10100
Группа №106	106.0	0_1101_10101

3.8. ВАРИАНТ_1. 2011 г. (“Архитектура ЭВМ и язык ассемблера”)

1. Ответ:
а) AL = 7C (16-ричн.), 124 (10-тичн. б/зн.), 124 (10-тичн. зн.)
CF = 1, SF = 0, OF = 1, ZF = 0.
б) AL = FA (16-ричн.), 250 (10-тичн. б/зн.), -6 (10-тичн. зн.)
CF = 1, SF = 1, OF = 0, ZF = 0.

2. Ответ:
EAX = 0xDDE

3. Решение:
```
MOV EAX, DWORD [p + 32]
MOV EAX, DWORD [EAX]
INC EAX
MOV DWORD [x], EAX
```

4. Решение:
```
static int a;
static int b;
static int c;
static int d;
if (a > b) {
   d = c / (a - b);
} else if (a < b) {
```

```
      d = c / (b - a);
   } else {
      d = 0;
   }
```

5. Решение:

```
section .bss
   a resd N

section .text
CMAIN:
   XOR ECX, ECX
   MOV EBX, N-1
.loop:
   MOV EAX, DWORD [A+4*ECX]
   CMP EAX, DWORD [A+4*EBX]
   JNE .set_false
   INC ECX
   DEC EBX
   CMP ECX, EBX
   JL .loop
   MOV EAX, 0
   JMP .end
.set_false:
   MOV EAX, 1
.end:
   RET
```

6. Ответ:

H = 4, J = 8

7. Решение:

```
int f(signed char y, short *b, int c, int x);
```

Помимо приведенного решения, для переменных b и x допустим тип unsigned; для переменной x – тип short

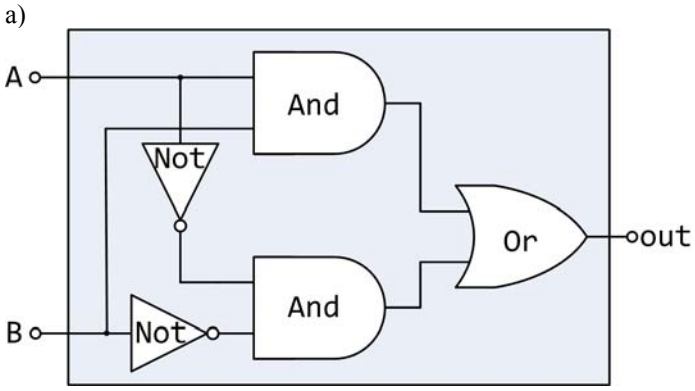
8. Решение:

```
CEXTERN malloc
seq:
   PUSH EBP
   MOV EBP, ESP
```

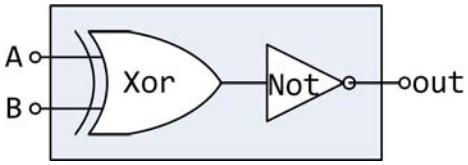
```
   SUB ESP, 8     ; общий размер фрейма 16 байт
   MOV EDX, DWORD [EBP+8]
   MOV EAX, EDX
   SAL EAX, 2
   MOV [ESP], EAX
   CALL MALLOC
   XOR ECX, ECX
.loop:
   MOV DWORD [EAX+4*ECX], ECX
   INC ECX
   CMP ECX, EDX
   JL .loop
   MOVE SP, EBP
   POP EBP
   RET
```

9. Ответы:

а)



Помимо того, допустима реализация сравнения двух битов с применением элемента Xor.



б) $T_{доступа} = T_{ср.\ поиск} + T_{ср.\ вращения} + T_{ср.\ передача}$

$T_{ср.\ вращения} = 1/2 \times 1/RPM \times 60\ с / 1\ мин.$

$T_{ср.\ передача}$ = 1/RPM x 1/(ср. # секторов на дорожке) x 60 с./1 мин.

$T_{доступа}$ = 12 + 5,(5) + 0,(037) = 17,593 мкс.

10. Ответы:

а) Препроцессор, компилятор, ассемблер, компоновщик.

б) Ошибка компоновки: два сильных символа f.
Запись в y (файл f2.c) меняет значение переменной y в файле f1.c.

СОДЕРЖАНИЕ