Глава 5. Коллоквиум, методика проведения и типовые вопросы.

Коллоквиум проводится на семинарском занятии в середине семестра (обычно – до 1-го ноября) в форме тотального опроса с билетами.

Билеты

1. Вариант 1) Основы синтаксиса языка Си. Ключевые слова. Фундаментальные типы данных. Определение переменных и констант. Выражения, операции, комментарии. 2) Разные виды цикла while, do.
2. Вариант 1) Операторы в выражениях языка Си. Приоритет операторов. Оператор sizeof(). 2) Директива #define препроцессора и ее использование. Макроопределения с параметром.
3. Вариант 1) Управляющий оператор switch. 2) Директивы условной компиляции препроцессора и их использование.
4. Вариант 1) Управляющие операторы if, goto. 2) Модульный подход в программировании. Использование *.h файлов. Раздельная компиляция.
5. Вариант 1) Массивы. Передача массивов в функции. 2) Локальные, глобальные, статические переменные.
6. Вариант 1) Основные действия, выполняемые при помощи отладчика в среде Microsoft Visual Studio. 2) Цикл for.
7. Вариант 1) Функция main(), ее параметры. 2) Оператор typedef. Приведение типов.
8. Вариант 1) Использование функций: заголовок, тело и вызов функции. 2) Управляющие операторы if, goto.
9. Вариант 1) Статические функции. 2) Блоки и правила видимости переменных.
10. Вариант 1) Передача параметров в функции. Передача параметров по значению. 2) Операторы в выражениях языка Си. Приоритет операторов. Оператор sizeof().
11. Вариант 1) Работа в среде Microsoft Visual Studio: создание проекта. Основные файлы проекта. Этапы создания программы: компиляция, линковка, запуск на выполнение, отладка. 2) Цикл for.
12. Вариант 1) Разные виды цикла while, do. 2) Массивы. Передача массивов в функции.
13. Вариант 1) Директива #include препроцессора и ее использование. 2) Функция main(), ее параметры.
14. Вариант 1) Процедурный подход программирования. Определение функции. Прототип функции. 2) Операторы инкремента и декремента.
15. Вариант 1) Передача параметров в функции. Передача параметров по значению. 2) Директивы условной компиляции препроцессора и их использование.

16. Вариант 1) Локальные, глобальные, статические переменные. 2) Использование функций: заголовок, тело и вызов функции.
17. Вариант 1) Директива #define препроцессора и ее использование. Макроопределения с параметром. 2) Управляющий оператор switch.
18. Вариант 1) Разные виды цикла while, do. 2) Директива #include препроцессора и ее использование.
19. Вариант 1) Работа в среде Microsoft Visual Studio: создание проекта. Основные файлы проекта. Этапы создания программы: компиляция, линковка, запуск на выполнение, отладка. 2) Глобальные и внешние переменные.
20. Вариант 1) Виды операторов присваивания в языке Си. 2) Оператор typedef. Приведение типов.
21. Вариант 1) Операторы инкремента и декремента. 2) Рекурсивный вызов функций.
22. Вариант 1) Операторы в выражениях языка Си. Приоритет операторов. Оператор sizeof(). 2) Статические переменные.
23. Вариант 1) Основные действия, выполняемые при помощи отладчика в среде Microsoft Visual Studio. 2) Массивы: одномерные и двумерные.
24. Вариант 1) Разные виды цикла while, do. 2) Модульный подход в программировании. Использование *.h файлов. Раздельная компиляция.
25. Вариант 1) Логические (булевские) операторы и операторы сравнения. 2) Цикл for.
26. Вариант 1) Массивы. Передача массивов в функции. 2) Локальные, глобальные, статические переменные.
27. Вариант 1) Основные действия, выполняемые при помощи отладчика в среде Microsoft Visual Studio. 2) Массивы: одномерные и двумерные.
28. Вариант 1) Функция main(), ее параметры. 2) Статические переменные.
29. Вариант 1) Операторы инкремента и декремента. 2) Управляющие операторы if, goto.
30. Вариант 1) Статические функции. 2) Управляющий оператор switch.

Факультативные задания

Так как коллоквиум проводится на семинарском занятии, то у студентов появляются дополнительные 2 недели, которые следует использовать для доделки и сдачи всех выданных ранее задач. Практика показывает, что к этому моменту «хвосты» имеет, как минимум, половина группы.

Студенты, которые успешно сдали все предыдущие задания, могут получить факультативное задание. Учет этих факультативных задач отличается от учета обычных заданий: они не являются обязательными, то есть, отсутствие их решения – не ухудшает оценки за успеваемость студента в течение семестра, в то же

время, наличие такой сданной задачи улучшает оценку студента. Если студент не успевает их сдать до шестого семинара, то такому студенту рекомендуется сначала сдавать плановые, обычные задания и лишь когда плановые задания сданы – вернуться к выполнению факультативных заданий.

Во всех приведенных ниже заданиях нужно оформить основную часть задания в виде функции, которой передаются все необходимые аргументы и которая возвращает результат. Весь ввод-вывод сосредоточить в функции `main()`.

1. Вариант Написать программу, вычисляющую наибольший общий делитель для двух введенных чисел. Алгоритм Евклида для нахождения наибольшего общего делителя двух чисел выглядит следующим образом. Даны два числа:a и b. Пока они не равны, вычитать из большего числа меньшее. Оставшееся значение a (или b) и будет наибольшим общим делителем двух чисел.

2. Вариант Лягушка изначально сидит в точке 0 числовой прямой. Каждую секунду она прыгает на 1 вправо, пока не достигнет точки K. Затем она начинает каждую секунду прыгать на 1 влево, пока не вернется в точку 0, затем – опять вправо и т. д. Написать программу определяющую, где окажется лягушка через T секунд. Значения K и T ввести с клавиатуры, если их нет в параметрах командной строки.

3. Вариант

Написать программу, вычисляющую определитель квадратной матрицы произвольной размерности.

Размерность и коэффициенты матрицы (не более 10) запрашиваются у пользователя, программа должна напечатать матрицу и ее определитель.

4. Вариант

Написать программу, выполняющую символьное умножение «в столбик» двух шестнадцатеричных чисел, вводимых пользователем «цифра за цифрой».

В программе использовать таблицу умножения для шестнадцатеричных цифр и правила умножения «в столбик». Не использовать приведение шестнадцатеричного числа к десятичной системе счисления.

5. Вариант

Написать программу, которая из данных n точек на плоскости находит все тройки точек, которые образуют равносторонние треугольники. Значения координат точек генерируются случайным образом в диапазоне [a, b], значения n, a, b вводятся пользователем.

6. Вариант

Отсортировать массив из уникальных элементов алгоритмом QuickSort: выбрать случайный элемент из середины массива, двигаясь от начала массива остановиться на элементе, который больше выбранного, двигаясь от конца массива остановиться на элементе, который меньше выбранного. Обменять найденные два элемента. Продолжить движение с обоих концов массива к его центру. Получив разбиение массива, в котором все элементы до некоторого – меньше выбранного, а все элементы после него – больше выбранного, повторить алгоритм рекурсивно для каждой из двух частей разбиения.

Сравнить количество операций сравнения и присваивания элементов массива с аналогичными показателями пузырьковой сортировки.

7. Вариант

Вычислить число π методом Монте-Карло: взять круг, вписанный в квадрат со стороной 10. Сгенерировать 1000 пар случайных чисел, каждое в диапазоне от 0 до 10. Каждую пару чисел считать `x-` и `y-` координатой некоторой точки внутри заданного квадрата. Для каждой такой точки определить – лежит ли она внутри круга? Отношение числа точек, попавших в круг к общему числу точек оценивает величину π / 4.

Сравнивая результат со встроенной константой `M_PI` оценить погрешность этого метода в зависимости от выбранного количества пар случайных чисел.

8. Вариант

Построить свой собственный генератор случайных чисел, основанный на линейном конгруэнтном методе: $x_{i+1} = (ax_i + c) \% M$ ($a < M$, $c < M$). Построить (в виде символьной матрицы 20*20) гистограмму достаточно большой сгенерированной последовательности чисел.

По гистограмме «на глаз» оценить равномерность распределения в зависимости от разных значений параметров `a, c, M`.

9. Вариант

Написать рекурсивную версию алгоритма, переводящего заданное число в двоичную систему счисления.

10. Вариант

Написать рекурсивную версию алгоритма, распечатывающего все простые множители заданного числа.

11. Вариант

В декартовой системе координат на плоскости в виде массива заданы координаты вершин треугольника. Определить, принадлежит ли точка с координатами (x,y) этому треугольнику.

12. Вариант

Даны N целых чисел X_1, X_2, ..., X_N. Требуется вычеркнуть из них минимальное количество чисел так, чтобы оставшиеся шли в порядке возрастания.

13. Вариант

Задан вес E пустой копилки и вес F копилки с монетами. В копилке могут находиться монеты N видов, для каждого вида известна ценность Pi и вес Wi одной монеты. Найти минимальную и максимальную суммы денег, которые могут находиться в копилке.

14. Вариант

Число называется совершенным, если оно равно сумме всех своих делителей, меньших его самого. Требуется найти все совершенные числа от M до N.

15. Вариант

Вывести все представления натурального числа N суммой натуральных чисел. Перестановка слагаемых нового способа представления не даёт.

16. Вариант

Многоугольник на плоскости задан целочисленными координатами своих N вершин в декартовой системе координат. Требуется найти площадь многоугольника. Стороны многоугольника не соприкасаются (за исключением соседних - в вершинах) и не пересекаются.

17. Вариант

Дано целое неотрицательное число в I-ричной системе счисления. Вывести это число в J-ричной системе счисления

18. Вариант

Дано N прямоугольников со сторонами, параллельными осям координат. Требуется определить площадь фигуры, образованной объединением данных прямоугольников

19. Вариант

Вывести в порядке возрастания все несократимые дроби, заключённые между 0 и 1, знаменатели которых не превышают N.

20. Вариант .

Дано N отрезков провода длиной L_1, L_2, ..., L_N сантиметров. Требуется с помощью разрезания получить из них K равных отрезков как можно большей длины, выражающейся целым числом сантиметров.

21. Вариант

По координатам вершин многоугольника требуется найти координаты его центра тяжести. Стороны многоугольника друг с другом не соприкасаются (за исключением соседних - в вершинах) и не пересекаются.

22. Вариант

Натуральное число представлено массивом десятичных цифр произвольной длины. Реализовать умножение «в столбик» двух таких чисел.

23. Вариант

Натуральное число представлено массивом десятичных цифр произвольной длины. Реализовать деление «в столбик» двух таких чисел, с выдачей результата в виде десятичной дроби (целая и дробные части отдельно) с заданной точностью.

24. Вариант

Дана последовательность целых чисел, удовлетворяющая условию Фибоначчи
$F_k = F_{k-1} + F_{k-2}$ для любого k>2.
Найти n-й член последовательности, если известны i-й и i+1-й члены, *i* может быть как больше, так и меньше *n*.

25. Вариант

В таблице из N строк и N столбцов клетки заполнены цифрами от 0 до 9. Требуется найти такой путь из клетки (1, 1) в клетку (N, N), чтобы сумма цифр в клетках, через которые он пролегает, была минимальной; из любой клетки ходить можно только вниз или вправо.
Выводятся *N* строк по *N* символов. Символ решётка показывает, что маршрут проходит через эту клетку, а минус - что не проходит. Если путей с минимальной суммой цифр несколько, вывести любой.

26. Вариант

Когда некоторая функция задана таблицей своих значений в большом количестве точек, то классическая интерполяция полиномами Лагранжа (см. курс лекций) будет давать неважные результаты. Вместо этого часто используют интерполяцию значений такой функции с помощью сплайнов: на каждом i-том интервале таблицы будем приближать заданную функцию полиномом порядка k (назовем его Sk(i)) так, чтобы выполнялось равенство значений двух соседних полиномов в каждом i-том узле таблицы Sk(i-1) = Sk(i) и значений всех производных этих двух соседних полиномов в этом же узле таблицы.
Набор этих условий дает однозначный алгоритм вычисления коэффициентов всех полиномов для всех интервалов, если на краях (то есть, в узлах таблицы 0 и N-1) мы будем полагать значения самых старших производных равными нулю. Для k=1 мы получим обычную кусочно-линейную интерполяцию, для больших k мы получим весьма гладкую, красиво выглядящую кривую с минимальными изгибами.
Постройте для заданной произвольной таблицы пар точек xi yi интерполяцию сплайнами второго, а если сможете – то и третьего порядка и сравните ее работу с интерполяцией полиномом Лагранжа.

27. Вариант

Для функции y=sin(1/x) в ближней окрестности нуля сложно применить обычные методы вычисления определенного интеграла, так как она очень быстро меняется и методы, использующие разбиение на равные интервалы будут работать плохо.

Попробуйте реализовать подсчет такого определенного интеграла, разложив эту функцию в ряд Тейлора и вычисляя сумму определенных интегралов для каждого члена этого разложения до достижения заданной точности. Точность можно контролировать как модуль разности между двумя последовательными приближениями значения вычисляемого интеграла.

28. Вариант

Максимально быстро найти все возможные корни уравнения $\cos(x) + \cos(x^2) - 1.55 = x$

Проблема в том, что метод касательных будет гарантировать сходимость в довольно узкой окрестности каждого корня, поэтому нужно использовать комбинацию метода дихотомии или хорд (чтобы приблизиться к корню) и переход к методу касательных, когда это становится возможно (т.е. выполняется условие сходимости).

29. Вариант

Алгоритм быстрой сортировки имеет оценку времени его работы пропорциональный $N*\log_2(N)$. Построим алгоритм сортировки натуральных чисел, который будет работать за константное время. Учтем, что в компьютерном представлении все натуральные числа имеют одно и то же количество двоичных разрядов (скажем, 32). Организуем два прохода нашего массива, движущихся навстречу друг другу, как в алгоритме QuickSort: только первый проход (идущий от начала массива) будет обнаруживать единицу в старшем разряде числа, а второй (идущий навстречу от конца массива) будет обнаруживать ноль в этом же старшем разряде числа. Как только такая пара чисел будет обнаружена – поменяем их местами и продолжим эти проходы. В конце проходы сойдутся где-то в середине массива, при этом все числа слева от этой точки будут иметь в старшем разряде ноль, а все числа справа – единицу. Дальше применим этот же алгоритм рекурсивно к каждой из двух получившихся частей массива для второго по старшинству двоичного разряда, потом для третьего и так далее. Таким образом в самом конце и получим полностью отсортированный массив. Оценка производительности при этом получится 32*N, так как количество разрядов числа – 32 и на каждой итерации мы перебираем все элементы массива. Очевидно, что при очень больших N этот результат будет эффективнее быстрой сортировки.

30. Вариант

Дан большой массив произвольных натуральных чисел (без повторов), нужно максимально быстро отвечать на вопрос: есть в нем заданное значение или нет? Если его отсортировать и потом использовать для поиска метод деления пополам, то время такого поиска будет пропорционально $\log_2(N)$.

Построим другой алгоритм, который нам обеспечит константное время поиска: заведем вспомогательный массив на 25% (или на 50% или на 100%) больший исходного. Пусть его длина выбрана равной M, а исходного массива – $N < M$. Некоторое значение (например, -1) будем использовать как маркер «пустой ячейки» в этом втором массиве. Для каждого очередного элемента входного массива вычислим хэш-функцию $h(x) = x \% M$; значение которой мы будем использовать как индекс во втором массиве: и по этому индексу сохраним во второй массив это очередное значение x из исходного массива. Если ячейка занята (то есть, по индексу $I=h(x)$ лежит неотрицательное значение), то заглянем в следующий по порядку элемент второго массива, если он свободен – сохраним значение туда, если нет – сдвинемся еще на одну позицию дальше. Если в таком поиске мы окажемся на последнем элементе второго массива и он занят – то просто «обернемся» в начало: продолжим поиск свободной ячейки начиная с первого элемента. Так как второй массив больше – то свободную ячейку мы найдем довольно быстро. Таким образом мы сохраним во втором массиве (хэш-таблице) все значения исходного массива. Поиск в этой хэш-таблице устроен аналогично: если мы ищем значение y, то вычисляем хэш-функцию от него $h(y)$ и с этим индексом обращаемся в хэш-таблицу. Если ячейка свободна – значит y в массиве нет, если занята – то мы сравниваем y с лежащим в хэш-таблице значением. Совпало – значит мы нашли искомое, не совпало – переходим к следующей ячейке (опять, с «оборотом» через последний элемент, если это необходимо).