Семинар #3: Массивы. Домашнее задание.

Эти задачи нужно оформить в соответствии с правилами оформления:
http://style.vdi.mipt.ru/CodeStyle.html и youtube.com/watch?v=NSNvfr_KpDc
В некоторых задачах потребуются входные файлы, которые можно найти по адресу:
github.com/v-biryukov/cs_mipt_faki/tree/master/term1/seminar03_array/homework/files

Сортировка пузырьком

- **Задача 1 – Сортировка пузырьком:** Алгоритм сортировки пузырьком по возрастанию заключается в следующем:
  - Идём циклом с итерациями по `i` от `0` до `n`.
    - Проходим по подмассиву `[0:n-i]` и проверяем все пары соседних элементов. Если они стоят неверно, то меняем их местами. Понятно, что в результате такого прохода в конце подмассива будет стоять самое большое число.
  - После `n` таких итераций все элементы массива точно будут отсортированы.
  - Возможно массив будет отсортирован и раньше, чем `n` итераций. Поэтому можно считать количество перестановок на каждой итерации и, если перестановок не было, то завершать сортировку.

  Простую визуализацию этой сортировки можно посмотреть по ссылке: youtube.com/watch?v=xli_FI7CuzA
  Напишите функцию `void bubble_sort(int* array, int n)`, реализующую эту сортировку. Протестируйте работу вашей функции в функции `main`.

- **Задача 2 – Рекурсивная сортировка пузырьком:** Алгоритм рекурсивной сортировки пузырьком по возрастанию подмассива `[lo:hi]` заключается в следующем:
  - Если `hi - lo <= 1`, то заканчиваем выходим из функции (`return;`).
  - Проходим по подмассиву `[lo:hi]` и проверяем все пары соседних элементов. Если они стоят неверно, то меняем их местами. Понятно, что в результате такого прохода в конце подмассива будет стоять самое большое число.
  - Рекурсивно повторяем предыдущие шаги для подмассива `[lo:hi-1]`.

  Напишите функцию `void bubble_sort_rec(int* array, int lo, int hi)`, реализующую эту сортировку. Протестируйте работу вашей функции в функции `main`. Для сортировки всего массива нужно просто вызвать эту функцию с аргументами `lo = 0` и `hi = n`.

Простая работа с файлами. Функции `fprintf` и `fscanf`

Для простейшей работы с файлами мы будем использовать функции `fscanf` и `fprintf`. Подробней файлы будем проходить позднее.
Пример записи в файл:

```
#include <stdio.h>
int main()
{
        // Открываем файл под названием result.txt ( "w" = write - открываем файл на запись )
        // Так как файл открывается на запись, то необязательно чтобы он существовал
        FILE* fout = fopen("hello.txt", "w");
        fprintf(fout, "Hello world of files\n");

        // Закрываем файл
        fclose(fout);
}
```

Пример чтения чисел из файла:

```
#include <stdio.h>
int main()
{
        // Открываем файл под названием numbers.txt ( "r" = read - открываем файл на чтение )
        FILE* fin = fopen("numbers.txt", "r");

        int n;
        int a[100];
        fscanf(fin, "%d", &n);
        for (int i = 0; i < n; i++)
                fscanf(fin, "%d", &a[i]);

        fclose(fin);
}
```

- **Задача 3 – Чтение/запись:** Создайте файл `input.txt` в котором будут храниться входные числа в следующем виде:

  ```
  15
  54 32 53 64 1 21 77 4 6 81 34 10 92 17 42
  ```

  Сначала идёт число n - количество чисел последовательности, а потом эти n чисел. Ваша задача - написать программу, которая будет считывать эти числа, возводить их в квадрат и записывать результат в файл `output.txt`.

- **Задача 4 – Чтение/запись сортировки:** В файле `numbers.txt` хранятся 10000 случайных чисел. Отсортируйте их с помощью сортировки пузырьком. Результат запишите в файл `sorted_numbers.txt`. (Смотрите пример решения похожей задачи ниже)

Бинарный поиск

- **Задача 5 – Бинарный поиск:** Написать `int binsearch(int* array, int lo, int hi, int x)`, которая будет искать элемент `x` в отсортированном по возрастанию подмассиве `array[lo:hi]` и возвращать индекс этого элемента. Если такого элемента нет, то функция должна вернуть `-1`. Если таких элементов несколько, то функция может вернуть индекс любого из них.
  Алгоритм бинарного поиска в подмассиве `[lo:hi]`, отсортированном по возрастанию:
  - Если `hi - lo <= 0`, то выходим из функции и возвращаем `-1`.
  - Иначе, берём средний элемент (элемент с индексом `mid = (hi + lo)/2`).
    * Если этот элемент равен искомому, то возвращаем его индекс.
    * Если этот элемент больше искомого, то *возвращаем* результат бинарного поиска на подмассиве `[lo:mid]` (рекурсивный вызов).
    * Если этот элемент меньше искомого, то *возвращаем* результат бинарного поиска на подмассиве `[mid+1:hi]` (рекурсивный вызов).

  Проверьте вашу функцию на отсортированном массиве из файла `numbers.txt`:

число	номер числа в отсортированном массиве из файла `numbers.txt`
3	0
20	5
470	146
1000	-1
4000	1201
32764	9998

Пример программы, которая считывает числа из файла `numbers.txt`, сортирует их и записывает в файл `sorted.txt`.

```
#include <stdio.h>
#define MAX 10000
// Функция, которая принимает массив и сортирует его сортировкой выбором
void selection_sort(int* array, int n)
{
        for (int i = 0; i < n; ++i)
        {
                // Для подмассива [i:n]
                // Находим минимальный элемент на [i:n]
                int min_index = i;
                for (int j = i + 1; j < n; ++j)
                {
                        if (array[j] < array[min_index])
                                min_index = j;
                }

                // Меняем i - й элемент и минимальный
                int temp = array[i];
                array[i] = array[min_index];
                array[min_index] = temp;
        }
}

int main()
{
        // Открываем файл под названием numbers.txt на чтение (read --> "r")
        FILE* infile = fopen("numbers.txt", "r");
        // Считываем количество элементов из файла
        int n;
        fscanf(infile, "%d", &n);

        // Считываем сами элементы из файла
        int numbers[MAX];
        for (int i = 0; i < n; ++i)
                fscanf(infile, "%d", &numbers[i]);
        // Закрываем файл
        fclose(infile);

        // Сортируем
        selection_sort(numbers, n);

        // Открываем файл под названием sorted.txt на запись (write --> "w")
        FILE* outfile = fopen("sorted.txt", "w");

        // Печатаем в файл
        for (int i = 0; i < n; ++i)
                fprintf(outfile, "%d ", numbers[i]);
        // Закрываем файл
        fclose(outfile);
}
```

Двумерные массивы

```c
#include <stdio.h>
// Зададим константу MAX = 200 - максимальный возможный размер матрицы
#define MAX 200

// В отличии от одномерного массива, в двумерном массиве при передаче
// в функцию обязательно нужно указывать размер ( количество столбцов = MAX )
// %g - печатает вещественные числа также как и %f, но без нулей на конце
void print_array(float arr[MAX][MAX], int n)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            printf("%5g ", arr[i][j]);
        printf("\n");
    }
}

// Суммируем 2 квадратные матрицы A и B размера n на n и записываем результат в C
void sum(float A[MAX][MAX], float B[MAX][MAX], float C[MAX][MAX], int n)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            C[i][j] = A[i][j] + B[i][j];
}
int main()
{
    // Создаём массивы вещественных чисел ( с запасом )
    float a[MAX][MAX] = {{7, 7, 2}, {1, 8, 3}, {2, 1, 6}};
    float b[MAX][MAX] = {{5, 2, 9}, {-4, 2, 11}, {7, 1, -5}};

    // Мы создали матрицы с 200 на 200 ( с запасом ), но будем использовать только
    // маленькую часть 3 на 3. В будущем мы научимся как создавать матрицы нужного
    // размера во время выполнения программы

    printf("a = \n");
    print_array(a, 3);

    printf("b = \n");
    print_array(b, 3);

    float c[MAX][MAX];
    sum(a, b, c, 3);
    printf("a + b = \n");
    print_array(c, 3);
}
```

- **Задача 6. Умножение на число:** Написать функцию `void multiply_by_number(float A[MAX][MAX], int n, float x)`, которая умножает квадратную матрицу `A` (`n x n`) на число `x`.
- **Задача 7. Присвоение:** Написать функцию `void assign(float A[MAX][MAX], float B[MAX][MAX], int n)`, которая присваивает элементам матрицы A соответствующие элементы матрицы `B` (`A = B`).
- **Задача 8. Умножение матриц:** Написать функцию `void multiply(float A[MAX][MAX], float B[MAX][MAX], float C[MAX][MAX]), int n`, которая перемножает матрицы `A` и `B` (строка на столбец), а результат записывает в матрицу `C`. Формула: $C_{ij} = \sum_k A_{ik} \cdot B_{kj}$.

  Проверьте ваш код на следующих тестах:

$$\begin{pmatrix} 7 & 7 & 2 \\ 1 & 8 & 3 \\ 2 & 1 & 6 \end{pmatrix} \cdot \begin{pmatrix} 5 & 2 & 9 \\ -4 & 2 & 11 \\ 7 & 1 & -5 \end{pmatrix} = \begin{pmatrix} 21 & 30 & 130 \\ -6 & 21 & 82 \\ 48 & 12 & -1 \end{pmatrix}$$

$$\begin{pmatrix} 5 & 2 & 9 \\ -4 & 2 & 11 \\ 7 & 1 & -5 \end{pmatrix} \cdot \begin{pmatrix} 7 & 7 & 2 \\ 1 & 8 & 3 \\ 2 & 1 & 6 \end{pmatrix} = \begin{pmatrix} 55 & 60 & 70 \\ -4 & -1 & 64 \\ 40 & 52 & -13 \end{pmatrix}$$

$$\begin{pmatrix} 7 & 7 & 2 \\ 1 & 8 & 3 \\ 2 & 1 & 6 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 7 & 7 \\ 3 & 8 & 1 \\ 6 & 1 & 2 \end{pmatrix}$$

  В файлах `mat_A10.txt` и `mat_B10.txt` лежат матрицы 10 на 10. считайте эти матрицы с помощью `fscanf`, перемножьте (A на B) и запишите результат в другой файл с помощью `fprintf`. В результате должно получиться:

$$\begin{pmatrix} 259 & -15 & 237 & 257 & 231 & 67 & 237 & -64 & 152 & 363 \\ 555 & 233 & 539 & 188 & 356 & 325 & 423 & -47 & 123 & 387 \\ 497 & 512 & 572 & 95 & 619 & 155 & 414 & 207 & 203 & 217 \\ 455 & 280 & 675 & 354 & 664 & 346 & 483 & 177 & 168 & 404 \\ 264 & 182 & 272 & 290 & 474 & -33 & 234 & 99 & 379 & 156 \\ 272 & 180 & 469 & 286 & 326 & 282 & 325 & 215 & 195 & 231 \\ 421 & 363 & 475 & 506 & 359 & 481 & 468 & 101 & 325 & 328 \\ 384 & 218 & 567 & 395 & 475 & 488 & 361 & 168 & 291 & 298 \\ 387 & 297 & 480 & 170 & 318 & 423 & 483 & 10 & -17 & 406 \\ 193 & 241 & 486 & 38 & 403 & 146 & 286 & 326 & 212 & 172 \end{pmatrix}$$

- **Задача 9. Матрица в степени:** Написать функцию `void power(float A[MAX][MAX], float C[MAX][MAX], int n, int k)`, которая вычисляет A^k, т.е. возводит матрицу A в k-ю степень, а результат записывает в матрицу C. Используйте функции `multiply` и `assign`. Псевдокод простейшей реализации такой функции:

```
float B[MAX][MAX]
B = A   // конечно нельзя приравнивать массивы, но можно использовать вашу функцию assign
C = A
for ( k - 1 раз )
{
        C = A*B
        B = C
}
```

  Проверьте ваш код на следующих тестах:

$$\begin{pmatrix} 7 & 7 & 2 \\ 1 & 8 & 3 \\ 2 & 1 & 6 \end{pmatrix}^4 = \begin{pmatrix} 7116 & 15654 & 9549 \\ 4002 & 8955 & 6135 \\ 3369 & 6165 & 4350 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}^{50} = \begin{pmatrix} 525456 & 396655 & 299426 \\ 696081 & 525456 & 396655 \\ 396655 & 299426 & 226030 \end{pmatrix}$$

- **Задача 10. Метод Гаусса:** Написать программу, которая бы решала линейную систему уравнений $Ax = b$ методом Гаусса. Главная функция этой программы должна иметь вид:
`void solve_linear_system(int n, float A[MAX][MAX], float b[], float x[])`.
Программа должна считывать матрицу A и столбец b из файла и записывать результат решения x в новый файл. Примерный вид вашей программы:

```
#include <stdio.h>
#define MAX 200

// Возможно понадобится вспомогательная функция для перестановки строк матрицы A
void swap_rows(float A[MAX][MAX], int n, int k, int m)
{
    // Ваш код
}

void solve_linear_system(float A[MAX][MAX], int n, float b[], float x[])
{
    // Ваш код
}

int main()
{
    int n;
    float A[MAX][MAX];
    float b[MAX];
    float x[MAX];

    // Считываем n, A и b из файла

    solve_linear_system(n, A, b, x);

    // Записываем x в новый файл
}
```

Проверьте вашу программу на следующих тестах:

1. Следующая система:
$$\begin{cases} x_1 + x_2 - x_3 = 9 \\ x_2 + 3x_3 = 3 \\ -x_1 - 2x_3 = 2 \end{cases}$$
Файл для считывания должен выглядеть следующим образом:
```
 3
 1 1 -1
 0 1  3
-1 0 -2
9
3
2
```
Решение этой системы: $x = (\frac{2}{3}, 7, -\frac{4}{3}) \approx (0.67, 7, -1.33)$
2. Система из файла `system1.txt`. Решение в файле `x1.txt`.
3. Система из файла `system2.txt`. Решение в файле `x2.txt`.