Непрямоугольные матрицы

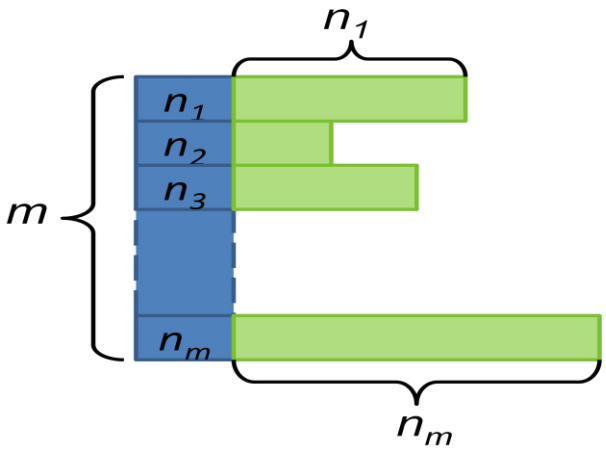
Здесь рассматриваются два аспекта:
1) как в физической памяти машины представлять непрямоугольные матрицы (структура данных, возможности языка программирования),
2) как работать с указателями (возможности языка программирования).

Задача 1

Из входного потока вводится непрямоугольная матрица вещественных чисел двойной точности.
В каждой строке матрицы найти максимальный элемент, а затем среди максимумов найти минимальный элемент. Результат вывести в выходной поток (на экран).

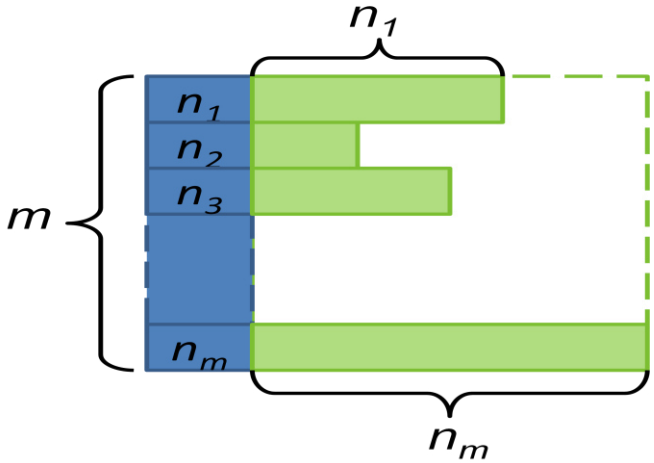
Задачу следует разбить на части, и каждую часть можно решать отдельно (независимо от других частей), а потом все соединить.
1. Исходная матрица непрямоугольная. Как ее представить в оперативной памяти? Какими средствами языка целесообразно пользоваться?
2. Каждая строка матрицы – одномерный массив (вектор) чисел. Какими средствами языка можно представить и обработать элементы массивов?

Начнем с представления непрямоугольной матрицы.
В соответствии с лекциями, непрямоугольная матрица может выглядеть так:



Соответственно, для каждой строки матрицы нужно иметь информацию о ее длине (т.е. количество элементов в каждой строке матрицы).
Как такую матрицу можно отобразить в памяти машины? Два способа.

– Статическое выделение памяти; при таком подходе определяется максимальная длина строки матрицы – *MaxN*, и под каждую строку выделяется одинаковое количество элементов:



В этом случае строку матрицы можно определить с помощью структуры:

```
struct Line{
      int n;            // количество элементов в строке матрицы
```

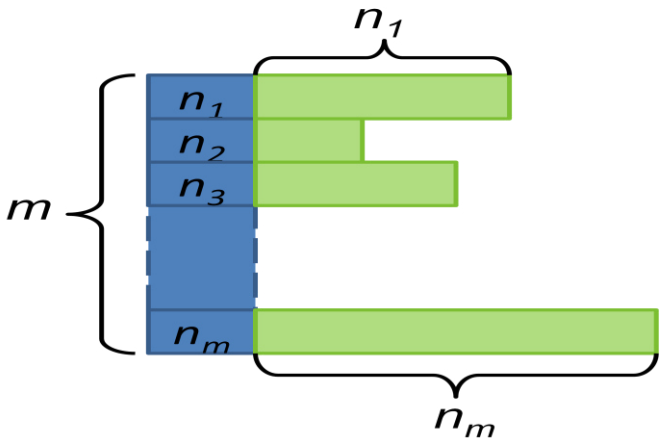
```
    double a[MaxN];   // массив элементов
};
```

Поскольку сама матрица может иметь произвольное количество строк, опять же определяется максимальное ожидаемое количество строк для выделения памяти – *MaxM*, но надо хранить и реальное количество строк. Поэтому для определения матрицы также целесообразно определить структуру:

```
struct Matrix{
    int lines;        // количество строк матрицы
    Line matr[MaxM]; // массив строк матрицы
};
```

При таком подходе выделяется (но не используется) лишняя память, причем память под матрицу выделяется на этапе компиляции программы и в дальнейшем, во время работы программы, изменена (перераспределена) быть не может.

– Динамическое выделение памяти; при таком подходе под каждую строку выделяется ровно столько памяти, сколько требуется:



В этом случае строку матрицы можно определить с помощью следующей структуры:

```
struct Line{
    int n;            // количество элементов в строке матрицы
    double *pa;       // указатель на первый элемент массива – динамический массив
};
```

Поскольку сама матрица также может иметь произвольное количество строк, для определения матрицы также целесообразно определить структуру:

```
struct Matrix{
    int lines;        // количество строк матрицы
    Line * pmatr;     // динамический массив строк матрицы
};
```

Для задания динамического массива используются соответствующие функции выделения памяти, которые выделяют требуемый объем памяти на этапе выполнения программы и возвращают значение указателя на выделенную память или `NULL`, если требуемый объем памяти не может быть выделен:

mun *ptr = (*mun* *)malloc(*количество_байтов*);

mun *ptr = (*mun* *)calloc(*количество_элементов*, *размер_элемента_в_байтах*);

Освобождение памяти выполняется с помощью следующей функции:

```
void free(void *);
```

Аргумент функции – значение указателя, полученное с помощью функций `malloc()` или `calloc()`.

Прототипы функций выделения и освобождения памяти находятся в файле `malloc.h`.

Рассмотрим функцию ввода такой непрямоугольной матрицы. Она использует рассмотренные ранее функции вода последовательности и целого числа, а также аналогичную функцию ввода вещественного числа двойной точности. Для проверки ввода матрицы рассмотрим и функцию вывода.

Текст программы:

```
/*****************************************************************************/
/* Из входного потока вводится матрица вещественных чисел двойной точности.  */
/* В каждой строке матрицы найти максимальный элемент,                        */
/* а затем среди найденных максимумов найти минимальный элемент.              */
/* Значение найденного минимального элемента вывести в выходной поток (на экран). */
/*                                                                            */
/* Примечание: матрица непрямоугольная, память выделяется динамически.        */
/* Для задания строки матрицы используется структура.                         */
/*****************************************************************************/

#include <stdio.h>
#include <malloc.h>

// структура для задания строки матрицы
typedef struct Line{
      int n; // количество элементов в строке матрицы
      double *a; // массив элементов
} Line;

// структура для задания самой матрицы
typedef struct Matrix{
      int lines; // количество строк матрицы
      Line *matr; // массив строк матрицы
} Matrix;

// прототипы функций
int getInt(int *); // ввод целого числа
int getDouble(double *); // ввод вещественного числа
int input(Matrix *a); // ввод матрицы
void output(const char *msg, Matrix a); // вывод матрицы
void erase(Matrix *a); // освобождение занятой памяти
double minmax(Matrix a); // вычисление главного результата
double max(double a[], int m); // вычисление max элемента вектора
double min(double a[], int m); // вычисление min элемента вектора
double mm(double a[], int m, int flag); // вариант - вычисление и max, и min

// основная функция
int main()
{
      Matrix matr = {0, NULL}; // исходный массив
      double res; // полученный результат

      if (input(&matr) == 0){ // ввод матрицы
            printf("%s\n", "End of file occured");
            return 1;
      }
      res = minmax(matr);  // вычисление требуемого результата
      output("Source matrix", matr);
      printf("Result: %f\n", res);
      erase(&matr); // освобождение памяти
      return 0;
```

```
}
// ввод целого числа
// приобнаружении ошибки ввод повторяется
// функция возвращает 1, если ввод корректен,
// и 0 при обнаружении конца файла
int getInt(int *a)
{
	int n;
	do{
		n = scanf_s("%d", a, sizeof(int));
		if (n < 0) // обнаружен конец файла
			return 0;
		if (n == 0){ // обнаружен некорректный символ - ошибка
			printf("%s\n", "Error! Repeat input");
			scanf_s("%*c", 0);
		}
	} while (n == 0);
	return 1;
}

// ввод вещественного числа
// приобнаружении ошибки ввод повторяется
// функция возвращает 1, если ввод корректен,
// и 0 при обнаружении конца файла
int getDouble(double *a)
{
	int n;
	do{
		n = scanf_s("%lf", a, sizeof(double));
		if (n < 0) // обнаружен конец файла
			return 0;
		if (n == 0){ // обнаружен некорректный символ - ошибка
			printf("%s\n", "Error! Repeat input");
			scanf_s("%*c", 0);
		}
	} while (n == 0);
	return 1;
}

// функция ввода матрицы
// функция возвращает 1, если ввод корректен, и 0 при обнаружении конца файла;
// если конец файла обнаружен в середине ввода - нужно освободить выделенную память
int input(Matrix *rm)
{
	const char *pr = ""; // будущее сообщение об ошибке
	int m;	// количество строк матрицы
	int i, j;
	double *p;
	// сначала вводим количество строк
	do{
		printf("%s\n", pr);
		printf("Enter number of lines: --> ");
```

```
            pr = "You are wrong; repeat, please!";
            if(getInt(&m) == 0)
                  return 0; // обнаружен конец файла; память не выделялась
      } while (m < 1);
      rm->lines = m;

      // выделяем память под массив структур - строк матрицы
      rm->matr = (Line *)calloc(m, sizeof(Line));

      for (i = 0; i < rm->lines; ++i){
            // теперь для каждой строки матрицы вводим количество столбцов
            pr = "";
            do{
                  printf("%s\n", pr);
                  printf("Enter number of items in line %d: --> ", i + 1);
                  pr = "You are wrong; repeat, please!";
                  if (getInt(&m) == 0){
                        // освобождение выделенной памяти
                        rm->lines = i; // сколько строк сформировано, память выделена
                        erase(rm);
                        return 0; // обнаружен конец файла
                  }
            } while (m < 1);
            rm->matr[i].n = m;
            // и выделяем необходимую память под элементы строки
            p = (double *)malloc(sizeof(double)* m);
            rm->matr[i].a = p;

            // теперь можно вводить сами элементы данной строки матрицы
            printf("Enter items for matrix line #%d:\n", i + 1);
            for (j = 0; j < m; ++j, ++p)
            if (getDouble(p) == 0){
                  // освобождение выделенной памяти
                  rm->lines = i + 1; // выделена память и под текущую, i-ю строку
                  erase(rm);
                  return 0;
            }
      }
      return 1;
}
// функция вывода
void output(const char *msg, Matrix a)
{
      int i, j;
      double *p;
      printf("%s:\n", msg);
      for (i = 0; i < a.lines; ++i){
            p = a.matr[i].a;
            for (j = 0; j < a.matr[i].n; ++j, ++p)
                  printf("%10lf ", *p);
            printf("\n");
      }
```

```
}

// функция освобождения занятой памяти
void erase(Matrix *a)
{
	int i;
	for (i = 0; i < a->lines; ++i)
		free(a->matr[i].a);
	free(a->matr);
	a->lines = 0;
	a->matr = NULL;
}

// функция вычисления главного результата
double minmax(Matrix pm)
{
	double *s = (double *)malloc(sizeof(double)* pm.lines); // вектор для
получения max элементов в строке - по строкам
	double res; // результат
	double *p = s;
	int i;
	for (i = 0; i < pm.lines; ++i)
		*p++ = max(pm.matr[i].a, pm.matr[i].n); // или s[i] = mm(pm.matr[i].a,
pm.matr[i].n, 1);
	res = min(s, pm.lines); // или res = mm(s, pm.lines, -1);
	free(s);
	return res;
}

// функция вычисления max элемента вектора
double max(double a[], int m)
{
	double res = *a; // предполагаемый max
	for (; m-- > 0; ++a)
	if (*a > res)
		res = *a;
	return res;
}

// функция вычисления min элемента вектора
double min(double a[], int m)
{
	double res = *a; // предполагаемый min
	for (; m-- > 0; ++a)
	if (*a < res)
		res = *a;
	return res;
}
```

```
// Универсальная функция;
// возвращает максимальный элемент, если flag = 1,
// и минимальный элемент, если flag = -1
double mm(double a[], int m, int flag)
{
      double res = *a;
      for (; m-- > 0; ++a)
            if (flag > 0 ? *a > res : *a < res) // или (*a * flag > res * flag)
                  res = *a;
      return res;
}
```