Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

КУРСОВАЯ РАБОТА

по курсу
"Информатика"
"Архитектура ЭВМ, системное программное обеспечение"
II семестр
Задание 7 "Разреженные матрицы"

Студент: Пашкевич А. Р.
Группа: 08-107, № по списку 10

Руководитель: Ридли М. К.
Ридли А. Н.

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2017

АННОТАЦИЯ

В данном документе описывается программа, написанная в соответствии с постановкой задачи на курсовое проектирование по теме "Разреженные матрицы" по дисциплине "Архитектура ЭВМ, системное программное обеспечение". Данная программа предназначена для обработки разреженных матриц с элементами комплексного типа. Входными данными является текстовый файл содержащий матрицу размером m × n. Для проверки работы программы разработан тестовый пример. Результаты тестирования доказывают, что программа правильно выполняет все операции по обработке входных данных и формирования выходных данных.

1. ПОСТАНОВКА ЗАДАЧИ

Составить программу на языке Си с функциями для обработки прямоугольных разреженных матриц с элементами **комплексного** типа, которая:

1. Вводит матрицы различного размера, представленные во входном текстовом файле в обычном формате (по строкам), с одновременным размещением ненулевых элементов в разреженной матрице в соответствии с заданной схемой;
2. Печатает введенные матрицы во внутреннем представлении согласно заданной схеме размещения и в обычном (естественном) виде;
3. Выполняет необходимые преобразования разреженных матриц (или вычисления над ними) путем обращения к соответствующим функциям;
4. Печатает результат преобразования (вычисления) согласно заданной схеме размещения и в обычном виде.

Для отладки использовать матрицы, содержащие 5-10% ненулевых элементов с максимальным числом элементов 100.

Схема размещения:

Схема размещения матрицы — **один вектор**, что соответствует варианту 2 задания.

Ненулевому элементу соответствуют две ячейки: первая содержит номер столбца, вторая содержит значение элемента. Нуль в первой ячейке означает конец строки, а вторая ячейка в этом случае содержит номер следующей хранимой строки. Нули в обеих ячейках являются признаком конца перечня ненулевых элементов разреженной матрицы.

0	row	col	value	col	value	...

...

0	row	col	value	...	0	0

Преобразование:

Вариант преобразования — 7. Найти строку, содержащую наибольшее количество ненулевых элементов, напечатать ее номер и сумму элементов этой строки. Если таких строк несколько, обработать все.

2. МЕТОД РЕАЛИЗАЦИИ

1) Создать набор тестовых файлов с прямоугольными матрицами с элементами комплексного типа.

2) Использовать команды обработки текстовых файлов ОС UNIX и переадресацию ввода-вывода.

3) Распечатать матрицу во внутреннем и внешнем представлении. Метод обхода заданной схемы размещения матрицы.

4) Найти строку, содержащую наибольшее количество ненулевых элементов, напечатать ее номер и сумму элементов этой строки.

3. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

Данная программа осуществляет ввод прямоугольной матрицы в обычном строчном представлении. После этого программа распечатывает внешнее и внутреннее представление матрицы, а затем выводит номер строки содержащей наибольшее количество ненулевых элементов, а также сумму элементов этой строки

4. ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММЕ

Необходимое программное и аппаратное обеспечение: компилятор gcc

Операционная система: GNU/Linux, UNIX, MS Windows

Язык: Си

Система программирования: Си

Способ вызова и загрузки: bash

4. ОРГАНИЗАЦИЯ ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ

Входные данные:

На первой строке входного файла — числа M и N — размеры матрицы, затем следуют MxN элементов матрицы.

Выходные данные:

Матрица во внутреннем представлении, матрица в обычном виде, результат вычисления функции.

5. ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ ДАННЫХ

```
typedef float _Complex compl_t;

typedef struct {
    int m;                // сколько строк
    int n;                // сколько столбцов
    int _size;            // счетчик элементов в data
    compl_t *_data;       // одномерный массив - комплексные числа
} matrix_t;
```

3. ОРГАНИЗАЦИЯ ИСХОДНОГО КОДА

Код организован следующим образом:
Управляющая программа, обеспечивающая ввод данных, реализована в **main.c**, а реализация структуры и функции с ней – в файлах **matrix.h** и **matrix.c**.
Вывод содержимого выходного файла и обработка данных реализована в файле **execute.c**.

Таблица 1: Структура файлов проекта:

matrix.h	Заголовочный файл с описанием структуры данных

main.c	Открытие входного файла. Создание матрицы m × n Печать матрицы во внутреннем представлении Печать матрицы в обычном представлении Печать номера строки содержащей наибольшее количество ненулевых элементов и элементов этой строки
matrix.c	Функции create, destroy, initiation (считывает и обрабатывает данные из файла), print и т. д.
makefile	Используется для управления сборкой проекта. Служит для сборки проекта и очистки каталогов проекта от образованных в процессе сборки файлов.

4. КАК ЭТО РАБОТАЕТ

Программа принимает на вход текстовый файл, считывает 2 числа: m — количество строк матрицы и n — количество стобцов матрицы. Создает пустую матрицу размером m × n.

Затем она считывает матрицу из входного файла и записывает ненулевые элементы матрицы в соответствии с заданной схемой в одномерный массив структуры: mat->_data[mat->_size-1] = complex:

0.0+0.0*I; 1.0+0.0*I; номер столбца; a+b*I; номер столбца; значение; ...; 0.0+0.0*I; 2.0+0.0*I; номер столбца; a+b*I; …; 0; 9.0+0.0*I; номер столбца; a+b*I; …; 0; 0.

После этого выводится внутреннее представление матрицы в виде вектора и матрица в обычном (естественном) виде, а также номер строки содержащей наибольшее количество ненулевых элементов и сумму элементов этой строки. После чего программа завершается.

void create(matrix_t *, int, int**)**

создает пустую матрицу m x n и выделяет память

void destroy(matrix_t ***)**

освобождение памяти от матрицы

void push_back(matrix_t *, float, compl_t**)**

добавляет в _data две "ячейки" col=nj value=c

void nextRow_set(matrix_t *, float**)**

добавляет в _data две "ячейки" col=0 value = row+1

void print_usual_view(matrix_t ***)**

выводит матрицу в обычном "квадратном" представлении m x n

void print_vector_view(matrix_t ***)**

выводит матрицу во "внутреннем представлении" т. е. один вектор

int get_first(matrix_t *, float**)**

возвращает индекс первой col!=0 в строке row или, если таковой нет то _size

compl_t get_el(matrix_t *, int, float**)**

возвращает значение данного k-ого элемента если таковой имеется или 0 (что равнозначно нулевому елементу), нужна для печати матрицы

int row_el_count(matrix_t *, float**)**

возвращает количество ненулевых элементов в строке

int max_count(matrix_t ***)**

ищет максимальное количество ненулевых элементов в строке по всей матрице

void initiation(FILE *, matrix_t *, int, int**)**

получает входной файл, матрицу и числа m и n — размеры матрицы

считывает с помощью функции **fscanf** отдельно действительную и мнимую части комплексного числа, а также знак перед мнимой частью, затем `c = pReal + pImag*I`

и если считанное число отлично от нуля запускает функцию **push_back**

void print_row_max_elem_count(matrix_t ***)**

выводит номер строки содержащей наибольшее количество ненулевых элементов и сумму элементов этой строки

```
$ ./matrix test1.dat

File test1.dat open for reading...

Entering matrix consists 9 rows and 11 columns...

One vector view:

|  0  |  1  ||  6  |  1.762-0.908i  ||  10  |  0.936+0.058i  ||  0  |  2  ||  2  |  24.216+0.349i  ||  5  |
 2.749+0.572i  ||  0  |  3  ||  8  |  11.432+0.563i
 ||  11  |  5.751-0.861i  ||  0  |  4  ||  2  |  6.337-0.394i  ||  3  |  11.052-0.156i  ||  6  |  52.430+0.884i
 ||  7  |  1.671+0.025i  ||  0  |  5  ||  0  |
 6  ||  7  |  7.552+0.783i  ||  8  |  4.912-0.808i  ||  0  |  7  ||  4  |  3.185-0.193i  ||  0  |  8  ||  0  |  9
 ||  1  |  3.525+0.183i  ||  2  |  5.524-0.1
32i  ||  0  |  0  |

|  0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 1.762-0.908i 0.000+0.000i 0.000+0.000i
|  0.000+0.000i 24.216+0.349i 0.000+0.000i 0.000+0.000i 2.749+0.572i 0.000+0.000i 0.000+0.000i 0.000+0.000i
|  0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 11.432+0.563i
|  0.000+0.000i 6.337-0.394i 11.052-0.156i 0.000+0.000i 0.000+0.000i 52.430+0.884i 1.671+0.025i 0.000+0.000i
|  0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i
|  0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 7.552+0.783i 4.912-0.808i
|  0.000+0.000i 0.000+0.000i 0.000+0.000i 3.185-0.193i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i
|  0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i
|  3.525+0.183i 5.524-0.132i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i 0.000+0.000i

ROW 4 containing the maximum number of elements (MAX=4), SUM of elements is equal to 71.490+0.359i
```

ЗАКЛЮЧЕНИЕ

Данная программа, составлена в соответствии с постановкой задачи на курсовое проектирование по теме "Разреженные матрицы" по дисциплине "Архитектура ЭВМ, системное программное обеспечение". При написании программы использованы методические указания по курсовому проектированию по дисциплине. Для проверки работоспособности программы и правильности обработки входных данных разработан тестовый пример. Тестирование программы подтвердило, что программа правильно выполняет обработку данных и выдаёт верные результаты. Всё это свидетельствует о работоспособности программы и позволяет сделать вывод о пригодности программы к решению практических задач по обработке разреженных матриц.

matrix.h

```
#ifndef __MATRIX_H__
#define __MATRIX_H__

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <complex.h>

typedef float _Complex compl_t;

typedef struct {
    int m; // сколько строк
    int n; // сколько столбцов
    int _size; // счетчик элементов в data
    compl_t *_data; //значение - комплексное число
} matrix_t;

void create(matrix_t *, int, int); // создает матрицу m x n пустую
void destroy(matrix_t *);
void push_back(matrix_t *, float, compl_t); // добавляет в _data две "ячейки" col=nj value=c
void nextRow_set(matrix_t *, float); // добавляет в _data две "ячейки" col=0 value=row+1
void print_usual_view(matrix_t *); // выводит матрицу в обычном квадратно-гнездовом представлении m x n
void print_vector_view(matrix_t *); // выводит матрицу во "внутреннем представлении" т е один вектор
int get_first(matrix_t *, float); //  возвращает первую col!=0 в строке row или если таковой нет _size
compl_t get_el(matrix_t *, int, float); // возвращает значение данного n-ного элемента если таковой имеется
или 0 (что равнозначно нулевому елементу)
int row_el_count(matrix_t *, float); // возвращает количество ненулевых элементов в строке
int max_count(matrix_t *);
void initiation(FILE *, matrix_t *, int, int);
void print_row_max_elem_count(matrix_t *);

#endif
```

matrix.c

```
#include "matrix.h"

void create(matrix_t *mat, int m, int n){
    mat->m = m;
    mat->n = n;
    mat->_size = 0;
    mat->_data = NULL;
    mat->_data = (compl_t *)malloc(1 * sizeof(compl_t)); //выделяем память
    mat->_data[0] = 0;
}

void destroy(matrix_t *mat){
    mat->_data = NULL;
    mat->_data = (compl_t *)malloc(1 * sizeof(compl_t));
    free(mat->_data);
}

void push_back(matrix_t *mat, float nj, compl_t c){
    mat->_size++;
    mat->_data[mat->_size-1] = nj;
    mat->_size++;
    mat->_data[mat->_size-1] = c;
}

void nextRow_set(matrix_t *mat, float mi){
    mat->_size++;
    mat->_data[mat->_size-1] = 0;
    mat->_size++;
    mat->_data[mat->_size-1] = mi;
}

void initiation(FILE *pfile, matrix_t *mat, int m, int n) {
    compl_t c;
    float pReal;
    float pImag;
    char sign;
    for (int row = 0; row < m; ++row) {
        // проверяем вдруг у нас предыдущая строка была совсем пустой тогда нам ее как бы и незачем хранить
        /*if (row!=0 && mat->_data[mat->_size-1]==0.0+0.0*I) {
         *           mat->_data[mat->_size] = row+1; // записываем на ее место значение+1 текущей строки
```

```
    } else {
        тут есть свои подводные камни - есть мнение что может не сработать функция get_el так как нужно,
        а ковыряться лень, пусть пока будут пустые строки
        */
        nextRow_set(mat, row+1); // иначе добавляем col=0 value=row+1
        //}
        for (int col = 0; col < n; ++col) {
            if (fscanf(pfile,"%f%c%fi",&pReal,&sign,&pImag)==3) {
                if( sign == '-' ) {
                    pImag *= -1;
                }
                c = pReal + pImag*I;
                if (c){
                    push_back(mat, col+1, c);
                }
            } else {
                printf("Failed to read complex.\n");
                exit(1);
            }
        }
    }
    nextRow_set(mat, 0); // иначе добавляем col=0 value=row+1
}
// выводим матрицу в традиционном виде
void print_usual_view(matrix_t *mat){
    for (int i = 0; i < mat->m; ++i) {
        int it = get_first(mat, i+1);
        printf("|  ");
        for (int j = 0; j < mat->n; ++j) {
            compl_t c = get_el(mat, it, j+1);
            printf("%.3f%+.3fi ",creal(c),cimag(c));
            if(c) it+=2; // если у нас вернулся не 0 то увеличиваем it на 2, т е перескакиваем текущую пару
col-value
        }
        printf(" |\n");
    }
}

// выводим матрицу как храним - в виде одного вектора
void print_vector_view(matrix_t *mat){
    for (int i = 0; i < mat->_size; ++i) {
        compl_t c = mat->_data[i];
        if (c == 0.0+0.0*I) {
            printf("|  0  |  %d  |", (int)creal(mat->_data[i+1]));
        } else {
            printf("|  %d  |  %.3f%+.3fi  |",(int)creal(c),creal(mat->_data[i+1]),cimag(mat->_data[i+1]));
        }
        i++;
    }
    printf("\n");
}

int get_first(matrix_t *mat, float row) {
    for (int j = 0; j < mat->_size; ++j) {
        if (mat->_data[j]==0.0+0.0*I && mat->_data[j+1]==row+0.0*I) {
            return j+2;
        }
    }
    return mat->_size;
}

compl_t get_el(matrix_t *mat, int it, float col) {
    if (it < mat->_size && col == (int)creal(mat->_data[it])) {
        return mat->_data[it+1];
    } else {
        return 0;
    }
}

int row_el_count(matrix_t *mat, float row) {
    int count = 0;
    int it = get_first(mat, row);
    while (mat->_data[it] != 0.0+0.0*I) {
        ++count;
        it+=2; // если у нас вернулся не 0 то увеличиваем it на 2, т е перескакиваем текущую пару col-value
    }
    return count;
}

int max_count(matrix_t *mat){
```

```
    int max = 0, tmp_max = 0;
    for (int i = 0; i < mat->m; ++i) {
        tmp_max = row_el_count(mat, i+1);
        if (tmp_max > max) max = tmp_max;
    }
    return max;
}

/*
 * Найти строку, содержащую наибольшее количество ненулевых элементов, напечатать ее номер и сумму элементов
этой строки.
 * Если таких строк несколько, обработать все.
 */
void print_row_max_elem_count(matrix_t *mat){
    int max = max_count(mat);
    compl_t sum = 0;
    for (int i = 0; i < mat->m; ++i) {
        int it = get_first(mat, i+1);
        int count = row_el_count(mat, i+1);
        if (count == max) {
            while (mat->_data[it] != 0.0+0.0*I) {
                sum += mat->_data[it+1];
                it+=2; // если у нас вернулся не 0 то увеличиваем it на 2, т е перескакиваем текущую пару
col-value
            }
            printf("\nROW %d containing the maximum number of elements ", i+1);
            printf("(MAX=%d), SUM of elements is equal to %.3f%+.3fi\n\n", max, creal(sum),cimag(sum));
        }
    }
}
```

main.c

```
#include "matrix.h"

int main(int argc, char *argv[]) {
    FILE *pfile = NULL;
    char *fname;
    int m = 0; // количество строк
    int n = 0; // количество столбцов
    matrix_t mat;

    if (argc == 2) {
        fname = argv[1];
    } else {
        printf("Invalid number of parameters\n");
        exit(1);
    }
    //Open the file for reading
    if(!(pfile = fopen(fname, "r"))) {
        printf("Error opening %s for reading. Program terminated.\n", fname);
        exit(1);
    } else printf("\nFile %s open for reading...\n", fname);

    // считываем количество строк и столбцов (размер матрицы) из файла
    if (fscanf(pfile, "%d %d", &m, &n)!=2) {
        printf("Error! Invalid input format! No number of rows specified. Program terminated.\n");
        exit(1);
    }
    printf("\nEntering matrix consists %d rows and %d columns...\n\n", m, n);
    create(&mat, m, n);
    initiation(pfile, &mat, m, n);
    fclose(pfile); // Close the file

    printf("One vector view:\n\n");
    print_vector_view(&mat);
    printf("\n\n");
    print_usual_view(&mat);

    print_row_max_elem_count(&mat);

    destroy(&mat);
    return 0;
}
```