

Wykonał: Radosław Smoter

Grupa: 14

Nr: 27

Numer zadania: 7

Przykład: 62

Prowadzący: Prof. dr hab. inż.
Volodymyr Samotyj

Politechnika Krakowska

Wydział Inżynierii Elektrycznej i Komputerowej

Sprawozdanie: Wstęp do Programowania

Spis treści

Polecenie.....	1
(1) Warunki sortowania:.....	1
Kod programu.....	2
Wyniki.....	6
Opis programu.....	7
Wnioski.....	8

Polecenie

Korzystając z funkcji `srand()` i `rand()` napisać kod programu generowania liczb pseudolosowych na podstawie, których zapisać do pliku tekstowego macierz liczb całkowitych $A = \{a_{ij}\}$ o wymiarach 10×10 . Maksymalna wartość $\max(a_{ij}) < 100$. Ciąg liczb pseudolosowych powinien być uzależniony od numeru zadania na liście. Następnie posortować macierz liczb całkowitych $A = \{a_{ij}\}$ metodą sortowania szybkiego i wyniki zapisać do pliku tekstowego.

(1) Warunki sortowania:

Posortować elementy kolumn od maksymalnego do minimalnego.

Kod programu

```
/*
 * Generate matrix 10x10 of random numbers;
 * Save it to a txt file
 * Sort the matrix regarding each column's elements' values, in descending order
 * Append sorted results to the file
 */

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

/* Generate random numbers from this seed. */
#define SEED 62

void randomNumbersMatrix(int [][][10]);
void saveMatrixToFile(int [][][10], char *, char *, char *);
void sortColumns(int [][][10]);

int main(void) {
    /* Matrix 10x10 of random numbers */
    int matrix[10][10];

    randomNumbersMatrix(matrix);

    /* Save matrix of random values */
    saveMatrixToFile(
        matrix,
        "results.txt",
        "w",
        "Matrix 10x10 of random numbers:\n");

    sortColumns(matrix);

    /* Save matrix of sorted values */
    saveMatrixToFile(
        matrix,
        "results.txt",
        "a",
        "Matrix 10x10 of numbers sorted in descending order:\n");

    return 0;
}

/* Generate a matrix filled with random numbers */
void randomNumbersMatrix(int matrix[][10]) {
    srand(SEED);

    for (int i = 0; i < 10; i++)
        for (int j = 0; j < 10; j++)
            matrix[i][j] = rand() % 100;
}
```

```
/* Saves specified array to a file. Takes the matrix, file name, mode in which file
should be saved (e.g. "r"), message to label the data in the file. */
```

```
void saveMatrixToFile(
    int matrix[][10],
    char *filename,
    char *mode,
    char *msg) {

    FILE *file = fopen(filename, mode);

    if (file != NULL) {
        /* Matrix entires */
        fprintf(file, "%s", msg);
        for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++)
                fprintf(file, "%6i", matrix[i][j]);
            fprintf(file, "\n");
        }
        fprintf(file, "\n");

        fclose(file);
    }
    else {
        fprintf(stderr, "Error: Can not open the file.");
        return;
    }
}
```

```
/* Exchange values of two variables of an array */
```

```
void exch(int v[], int n, int m) {
    int temp = v[n];
    v[n] = v[m];
    v[m] = temp;
}
```

```
/* Return true when first number is greater than the second */
```

```
short compare(int a, int b) {
    return (a - b) > 0; // True when a > b
}
```

```
/*
```

```
 * Logic for quicksort. Performs exchanges in two cases: value on the side of the
pivot, where it should be smaller than the pivot is higher or vice versa; or new pivot
is exchanged with the previous one (because it hasn't been compared yet).
```

```
 * Parameters: array to sort, index of a value that should be smaller, index of value
that should be higher
```

```
*/
```

```
int partition(int v[], int lo, int hi) {
```

```
    /* Index of an element with higher value than pivot, on the side, where elements'
values are ought to be lower */
```

```
    int i = lo;
```

```

/* Index of an element with lower value than pivot, on the side, where elements'
values are ought to be higher */
int j = hi+1;
/* Pivot */
int item = v[lo];

while (1) {
    /* Search for the i */
    while (compare(v[++i], item)) if (i == hi) break;
    /* Search for the j */
    while (compare(item, v[--j])) if (j == lo) break;
    /* End, when indexes meet */
    if (i >= j) break;
    /* Exchange values of elements standing on the wrong side of the pivot */
    exch(v, i, j);
}

/* Exchange pivot with an element with index j */
exch(v, lo, j);
/* New pivot */
return j;
}

/* Sort an array in descending order; takes array, lowest and highest indexes of the
subarray that should be sorted. */
void quicksort(int v[], int lo, int hi) {
    /* Base case */
    if (hi <= lo) return;
    /* New pivot */
    int j = partition(v, lo, hi);
    /* Sort left side */
    quicksort(v, lo, j-1);
    /* Sort right side */
    quicksort(v, lo+1, hi);
}

/* Sort the matrix regarding each column's elements' values, in descending order */
void sortColumns(int matrix[][10]) {
    /* Column */
    int col[10];

    for (int i = 0; i < 10; i++) {
        /* Index for the column each entry */
        int k = 0;

        /* Extract single column */
        for (int j = 0; j < 10; j++)
            /* Write from matrix to col */
            col[k++] = matrix[j][i];

        quicksort(col, 0, 9);

        /* Reset k */
        k = 0;
        for (int j = 0; j < 10; j++)

```

```
        /* Write from column to matrix */  
        matrix[j][i] = col[k++];  
    }  
}
```

Wyniki

Matrix 10x10 of random numbers:

74	58	2	44	97	83	27	74	81	55
81	75	29	56	7	86	36	48	98	60
60	39	4	20	32	20	4	27	69	88
68	95	46	22	91	95	6	71	22	87
26	3	62	7	11	21	93	0	69	91
60	81	83	65	53	15	85	58	42	6
46	63	2	44	85	93	91	43	64	13
83	42	16	97	49	80	19	42	80	40
86	40	22	69	57	75	36	43	85	31
1	31	46	3	27	83	49	19	27	65

Matrix 10x10 of numbers sorted in descending order:

86	95	83	97	97	95	93	74	98	91
83	81	62	69	91	93	91	71	85	88
81	75	46	65	85	86	85	58	81	87
74	63	46	56	57	83	49	48	80	65
68	58	29	44	53	83	36	43	69	60
60	42	22	44	49	80	36	43	69	55
60	40	16	22	32	75	27	42	64	40
46	39	4	20	27	21	19	27	42	31
26	31	2	7	11	20	6	19	27	13
1	3	2	3	7	15	4	0	22	6

Opis programu

Program ma w swojej funkcji `main()` zadeklarowaną tablicę `matrix[][]`. Funkcja `randomNumbersMatrix()` bierze `matrix[][]` jako parametr i zapisuje do niej wygenerowane liczby losowe. Następnie `saveMatrixToFile()` zapisuje `matrix[][]` do pliku tekstowego „results.txt” w trybie zapisu „w”, z opisem „Matrix 10x10 of random numbers:\n”. Dalej, `sortColumns()` sortuje `matrix[][]` pod względem wartości kolumn, w kolejności malejącej. Posortowaną tablicę `matrix[][]`, funkcja `saveMatrixToFile()`, zapisuje do pliku „results.txt”, w trybie zapisu rozszerzającego plik „a”, z opisem „Matrix 10x10 of numbers sorted in descending order:\n”.

Funkcja `randomNumbersMatrix()` przyjmuje jako argument tablicę `matrix[][]` i zapisuje do niej liczby losowe wygenerowane z ziarna SEED, określonego w makrze.

Funkcja `saveMatrixToFile()` przyjmuje w swoich parametrach macierz, która zostanie zapisana, nazwę pliku, do którego odbędzie się zapis, tryb zapisu („w”, „a”) oraz etykietę, do opisanie zawartości zapisywanej tablicy. W razie wystąpienia błędu otwarcia pliku o podanej nazwie, zwracany jest błąd oraz kończona funkcja.

Funkcja `sortColumns()` iteruje po wszystkich kolumnach tablicy `matrix[][]`. Do tablicy `col[]` zapisuje każdą kolumnę, którą potem przekazuje funkcji `quicksort()`. Następnie `col[]` jest nadpisywane do `matrix[][]`.

Funkcja `quicksort()` w sposób rekurencyjny sortuje tablicę `v[]` dostarczoną jej w parametrze. Kolejnymi jej parametrami są najniższy `lo` i najwyższy `hi` indeks tej tablicy. Funkcja kończy działanie jeśli `hi <= lo`. W przeciwnym razie, wywołuje się funkcja `partition()`, która przeprowadza sortowanie oraz zwraca w swoim wyniku numer nowego piwołu. Następnie rekurencyjnie sortowane są odpowiednio lewa, a następnie prawa strona od piwołu w tablicy `v[]`.

Funkcja `partition()` przyjmuje takie same parametry jak `quicksort()`. Ta funkcja odpowiada za zamiany dokonywane w `v[]`. Jeżeli wartość tablicy `v[]` po stronie piwołu, gdzie wartości powinny być większe od piwołu, jest od niego mniejsza, to dokonuje się zmiana tego elementu z wartością po lewej stronie piwołu z elementem, którego wartość powinna być mniejsza od piwołu, a jest większa. To działanie wykonywane w pętli `while()`. Poza pętlą odbywa się kolejne wywołanie funkcji dokonującej zmianę pozycji elementów. Dotyczy ono samego piwołu, który zamieniany jest pozycją z nowym piwołem, wartością `j`. Wartością zwracaną przez funkcję jest nowy piwot, `j`.

Wnioski