****

**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII**

**AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**Calancea Cătălin**

**MI-222**

**Raport**

**pentru lucrarea de laborator Nr.3**

***la cursul de “Structuri de date şi algoritmi”***

Verificat:

**Guțu Maria,** *doctor, conf. univ.*

Departamentul Informatică şi IS,

Facultatea FCIM, UTM

**Chișinău – 2023**

**Scopul lucrării:** Programarea algoritmilor de prelucrare a tablourilor aplicând tehnicile și metodele de sortare prin utilizarea funcțiilor, pointerilor, alocării dinamice a memoriei în limbajul C.

**Nr. variantei:** 4

**Condiția problemei ( sarcinii de lucru ) :**

Elaborați un program C care va crea un meniu recursiv. Acesta trebuie să cuprindă următoarele funcții în C (cu apelare ulterioară ale acestora în funcția *main()*):

*1. Introducerea valorilor tabloului de la tastatură;  
2. Completarea tabloului cu valori random;  
3. Afișarea elementelor tabloului la ecran;  
4. Sortarea elementelor tabloului conform variantelor:*

1) Bubble Sort;   
2) Selection Sort;   
3) Insertion Sort;   
4) Merge Sort;  
5) Quick Sort;  
6) Shell Sort;  
7) Counting Sort;   
8) Heap Sort;  
9) Radix Sort;   
10) Comb Sort;

*5. Eliberarea memoriei și ieșirea din program.*

1. Se dă un array unidimensional cu elemente de tip *integer* și un număr natural *n*, valoarea căruia este citită de la tastatură. Să se afișeze array-ul original și array-ul modificat după fiecare manipulare a datelor din array, afișarea rezultatelor ca și concluzie (De ex.: Nu există numere prime; Nu există așa element cu indexul dat și alte date stipulate în condiția problemei) etc.

A. Dacă media aritmetică a elementelor de pe poziții pare este mai mare decât media aritmetică a elementelor de pe pozițiile impare, atunci să se sorteze elementele array-ului crescător, aplicând tehnica de sortare HeapSort, altfel să se sorteze elementele array-ului descrescător, aplicând tehnica de sortare CountingSort.

B. Dacă există numere prime în array, atunci să se sorteze elementele array-ului crescător, aplicând tehnica de sortare RadixSort, altfel să se sorteze elementele array-ului descrescător, aplicând tehnica de sortare CombSort.

C. Dacă produsul elementelor negative este un număr negativ (array va conține atât elemente pozitive, cât și elemente negative), atunci să se sorteze elementele array- ului descrescător, aplicând tehnica de sortare MergeSort, altfel să se sorteze elementele array-ului crescător, aplicând tehnica de sortare BubbleSort.

**Codul programului:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

void checkA(int \*vect, int n);

void checkB(int \*vect, int n);

void checkC(int \*vect, int n);

int divnum(int x);

void menu(int \*vect, int n);

void allocate(int \*vect, int n);

void input(int \*vect, int n);

void display(int \*vect, int n);

void freeMem(int \*vect);

void swap(int \*a, int \*b);

void heapSort(int \*vect, int n);

void countingSort(int \*vect, int n);

void radixSort(int \*vect, int n);

void countingSortForRadix(int \*vect, int n, int place);

void combSort(int \*vect, int n);

int newGap(int gap);

void mergeSort(int \*vect, int l, int r);

void bubbleSort(int \*vect, int n);

int main(){

int \*vect = NULL, n;

printf("Introduceti n:\n");

scanf("%d", &n);

menu(vect, n);

return 0;

}

void menu(int \*vect, int n){

int option;

printf("Introduceti optiunea dorita:\n"

"1. Alocare memorie\n"

"2. Introducere in vector\n"

"3. Afisare vector\n"

"4. A\n"

"5. B\n"

"6. C\n"

"0. Eliberare memorie si iesire din program\n"

);

scanf("%d", &option);

switch(option){

case 1: allocate(vect, n); break;

case 2: input(vect, n); menu(vect, n); break;

case 3: display(vect, n); menu(vect, n); break;

case 4: checkA(vect, n); menu(vect, n); break;

case 5: checkB(vect, n); menu(vect, n); break;

case 6: checkC(vect, n); menu(vect, n); break;

case 0: freeMem(vect); break;

}

}

void checkA(int \*vect, int n){

int sumPare = 0, countPare = 0, sumImpare = 0, countImpare = 0;

for(int i = 0; i < n; i++){

if(i % 2 == 1){

sumPare += \*(vect + i);

countPare++;

}

else {

sumImpare += \*(vect + i);

countImpare++;

}

}

float avgPare = (float)sumPare / countPare;

float avgImpare = (float)sumImpare / countImpare;

if(avgPare > avgImpare) heapSort(vect, n);

else countingSort(vect, n);

}

int divnum(int x){

int count = 0;

for(int i = 2; i < x; i++){

if(x % i == 0){

count++;

break;

}

}

return count;

}

void checkB(int \*vect, int n){

int control = -1;

for(int i = 0; i < n; i++){

if(divnum(\*vect + i) == 1){

radixSort(vect, n);

control = 1;

break;

}

}

if(control == -1)

combSort(vect, n); //descr

}

void checkC(int \*vect, int n){

float prod = 1;

for(int i = 0; i < n; i++){

if(\*(vect + i) < 0) prod \*= \*(vect + i);

}

if(prod < 0) mergeSort(vect, 0, n - 1); //descr

else bubbleSort(vect, n);

}

void allocate(int \*vect, int n){

vect = malloc(n \* sizeof(int \*));

if(vect == NULL) printf("Nu s-a alocat memorie\n");

else printf("S-a alocat memorie\n");

menu(vect, n);

}

void input(int \*vect, int n){

for(int i = 0; i < n; i++){

scanf("%d", (vect + i));

}

}

void display(int \*vect, int n ){

for(int i = 0; i < n; i++){

printf("%d ", \*(vect + i));

}

printf("\n");

}

void swap(int \*a, int \*b){

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void heapify(int \*vect, int n, int i){

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if(left < n && \*(vect + left) > \*(vect + largest))

largest = left;

if(right < n && \*(vect + right) > \*(vect + largest))

largest = right;

if(largest != i){

swap((vect + i), (vect + largest));

heapify(vect, n, largest);

}

}

void heapSort(int \*vect, int n){

for(int i = n / 2 - 1; i >= 0; i--)

heapify(vect, n, i);

for(int i = 0; i >= 0; i--){

swap((vect), (vect + i));

heapify(vect, i, 0);

}

printf("Vectorul a fost sortat prin heapSort\n");

}

void countingSort(int \*vect, int n){

int \*tempVect = malloc(n \* sizeof(int \*));

int max = \*(vect);

for(int i = 1; i < n; i++){

if(max < \*(vect + i))

max = \*(vect + i);

}

int \*count = calloc(n, sizeof(int \*));

for(int i = 0; i < n; i++)

count[\*(vect + i)]++;

for(int i = 0; i < n; i++)

\*(count + i) += \*(count + i - 1);

/////////////////// = n - 1

for(int i = 0; i < n; i++){

tempVect[count[\*(vect + i) - 1]] = \*(vect + i);

count[\*(vect + i)] --;

}

for(int i = 0; i < n; i++)

\*(vect + i) = \*(tempVect + i);

printf("Vectorul a fost sortat prin countingSort\n");

}

int getMax(int \*vect, int n) {

int max = \*(vect);

for (int i = 1; i < n; i++)

if (\*(vect + i) > max)

max = \*(vect + i);

return max;

}

void radixSort(int \*vect, int n) {

int max = getMax(vect, n);

for (int i = 1; max / i > 0; i \*= 10)

countingSortForRadix(vect, n, i);

printf("Vectorul a fost sortat prin radixSort\n");

}

void countingSortForRadix(int \*vect, int n, int place) {

int output[n + 1];

int max = (\*vect / place) % 10;

for (int i = 1; i < n; i++) {

if (((\*(vect + i) / place) % 10) > max)

max = \*(vect + i);

}

int count[max + 1];

for (int i = 0; i < max; ++i)

count[i] = 0;

for (int i = 0; i < n; i++)

count[(\*(vect + i) / place) % 10]++;

for (int i = 1; i < 10; i++)

count[i] += count[i - 1];

for (int i = n - 1; i >= 0; i--) {

output[count[(\*(vect + i) / place) % 10] - 1] = \*(vect + i);

count[(\*(vect + i) / place) % 10]--;

}

for (int i = 0; i < n; i++)

\*(vect + i) = output[i];

}

void merge(int \*vect, int p, int q, int r) {

int n1 = q - p + 1;

int n2 = r - q;

int L[n1], M[n2];

for (int i = 0; i < n1; i++)

L[i] = vect[p + i];

for (int j = 0; j < n2; j++)

M[j] = vect[q + 1 + j];

int i, j, k;

i = 0;

j = 0;

k = p;

while (i < n1 && j < n2) {

if (L[i] <= M[j]) {

vect[k] = L[i];

i++;

} else {

vect[k] = M[j];

j++;

}

k++;

}

while (i < n1) {

vect[k] = L[i];

i++;

k++;

}

while (j < n2) {

vect[k] = M[j];

j++;

k++;

}

}

void mergeSort(int \*vect, int l, int r) {

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(vect, l, m);

mergeSort(vect, m + 1, r);

merge(vect, l, m, r);

}

printf("Vectorul a fost sortat prin mergeSort\n");

}

int newGap(int gap)

{

gap = (gap \* 10) / 13;

if (gap == 9 || gap == 10)

gap = 11;

if (gap < 1)

gap = 1;

return gap;

}

void combSort(int \*vect, int n)

{

int gap = n;

int temp, i;

for (;;)

{

gap = newGap(gap);

int swapped = 0;

for (i = 0; i < n - gap; i++)

{

int j = i + gap;

if (vect[i] < vect[j])

{

temp = vect[i];

vect[i] = vect[j];

vect[j] = temp;

swapped = 1;

}

}

if (gap == 1 && !swapped)

break;

}

printf("Vectorul a fost sortat prin combSort\n");

}

void freeMem(int \*vect){

free(vect);

printf("Memoria a fost eliberata\n");

exit(1);

}

void bubbleSort(int \*vect, int n){

bool sorted;

do{

sorted = true;

for(int i = 0; i < n - 1; i++){

if(\*(vect + i) > \*(vect + i + 1)){

int temp = \*(vect + i);

\*(vect + i) = \*(vect + i + 1);

\*(vect + i + 1) = temp;

sorted = false;

}

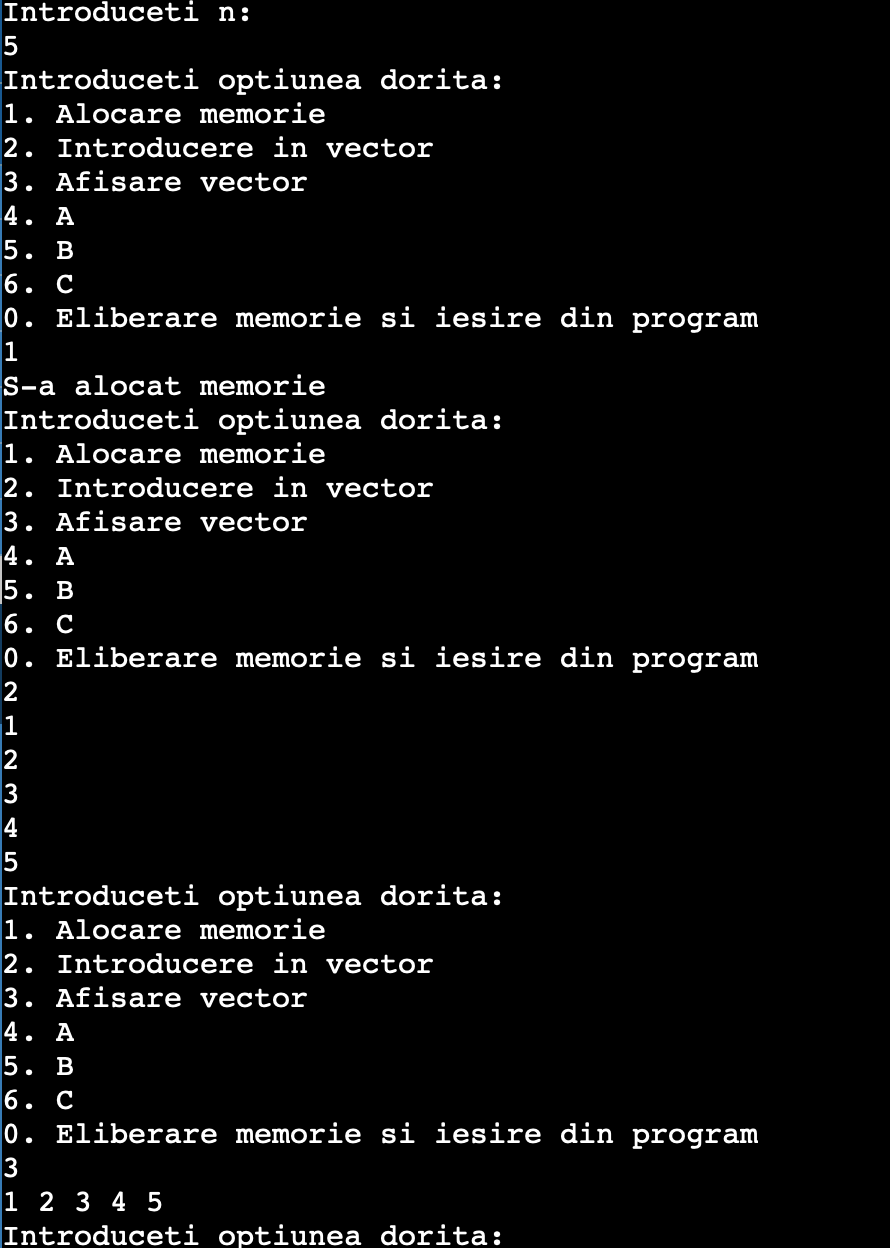
}

}

while(!sorted);

printf("Vectul a fost sortat prin bubbleSort\n");

}



**Concluzie:**În urma executarii lucrării de laborator nr.3 , am constatat ca programarea algoritmilor de prelucrare a tablourilor prin sortare este o parte importantă a programării în limbajul C și implică utilizarea unor tehnici și metode specifice. Utilizarea funcțiilor permite împărțirea algoritmilor în părți mai mici și mai ușor de gestionat, ceea ce face procesul de dezvoltare și de depanare mult mai ușor.