**.1SPRAWOZDANIE**

|  |  |  |  |
| --- | --- | --- | --- |
| **Przedmiot** | Wprowadzenie do Informatyki | **Zadanie** | 9.4 |
| **Autor** | Andrii Godlevskyi | **Grupa** | WCY23KY1S1 |
| **Temat** | Pliki i strumienie | | |

1.Treść zadania

Zmodyfikuj program z punktu 3, tak, aby łączył dane   
z 4 plików.

Metoda realizacji

Tworzymy funkcje porównania rekordów z plików wejściowych do sortowania, dalej sortujemy i wynik zapisujemy do stworzonego pliku i wypisujemy na ekranie.

1. Założenia / ograniczenia dotyczące danych:

1. Dane wejściowe

nazwa\_pliku we1, nazwa\_pliku we2, nazwa\_pliku we3, nazwa\_pliku we4, nazwa\_pliku wy – wczytywana z klawiatury

1. Dane wyjściowe

Dane z pliku we1, dane z pliku we2, dane z pliku we3,   
dane z pliku we4 - wyświetlone na ekranie.

Dane z pliku wy - zapisywane do pliku i wyświetlone na ekranie

1. Realizacja
2. Algorytm

Изображение выглядит как текст, рукописный текст, Бумажное изделие, бумага

Автоматически созданное описание



1. Kod zródłowy

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define RECORD\_SIZE 10 // Rozmiar rekordu

#define RECORDS\_PER\_LINE 10 // Ilosc rekordow w wierszu

// Deklaracja rekordu

typedef struct {

int data;

char character;

} Record;

// Funkcja do porownywania rekordow do sortowania

int compareRecords(const void \*a, const void \*b) {

Record \*recordA = (Record \*)a;

Record \*recordB = (Record \*)b;

return recordA->data - recordB->data;

}

int main() {

char filenameIn1[100], filenameIn2[100], filenameIn3[100], filenameIn4[100], filenameOut[100];

int n1, n2, n3, n4, i, j, k, l, m;

// Wczytanie nazw plikow

printf("Podaj nazwe pliku wejsciowego 1 (\*.dat): ");

scanf("%s", filenameIn1);

printf("Podaj nazwe pliku wejsciowego 2 (\*.dat): ");

scanf("%s", filenameIn2);

printf("Podaj nazwe pliku wejsciowego 3 (\*.dat): ");

scanf("%s", filenameIn3);

printf("Podaj nazwe pliku wejsciowego 4 (\*.dat): ");

scanf("%s", filenameIn4);

printf("Podaj nazwe pliku wyjsciowego (\*.dat): ");

scanf("%s", filenameOut);

// Otwarcie plikow wejsciowych

FILE \*fpIn1 = fopen(filenameIn1, "rb");

if (fpIn1 == NULL) {

printf("Blad otwarcia pliku wejsciowego 1.\n");

return 1;

}

FILE \*fpIn2 = fopen(filenameIn2, "rb");

if (fpIn2 == NULL) {

printf("Blad otwarcia pliku wejsciowego 2.\n");

fclose(fpIn1);

return 1;

}

FILE \*fpIn3 = fopen(filenameIn3, "rb");

if (fpIn3 == NULL) {

printf("Blad otwarcia pliku wejsciowego 3.\n");

fclose(fpIn1);

fclose(fpIn2);

return 1;

}

FILE \*fpIn4 = fopen(filenameIn4, "rb");

if (fpIn4 == NULL) {

printf("Blad otwarcia pliku wejsciowego 4.\n");

fclose(fpIn1);

fclose(fpIn2);

fclose(fpIn3);

return 1;

}

// Okreslenie liczby rekordow w każdym pliku

fseek(fpIn1, 0, SEEK\_END);

n1 = ftell(fpIn1) / sizeof(Record);

rewind(fpIn1);

fseek(fpIn2, 0, SEEK\_END);

n2 = ftell(fpIn2) / sizeof(Record);

rewind(fpIn2);

fseek(fpIn3, 0, SEEK\_END);

n3 = ftell(fpIn3) / sizeof(Record);

rewind(fpIn3);

fseek(fpIn4, 0, SEEK\_END);

n4 = ftell(fpIn4) / sizeof(Record);

rewind(fpIn4);

// Alokacja pamieci dla tablic rekordow

Record \*records1 = (Record \*)malloc(n1 \* sizeof(Record));

Record \*records2 = (Record \*)malloc(n2 \* sizeof(Record));

Record \*records3 = (Record \*)malloc(n3 \* sizeof(Record));

Record \*records4 = (Record \*)malloc(n4 \* sizeof(Record));

if (records1 == NULL || records2 == NULL || records3 == NULL || records4 == NULL) {

printf("Blad alokacji pamieci.\n");

fclose(fpIn1);

fclose(fpIn2);

fclose(fpIn3);

fclose(fpIn4);

free(records1);

free(records2);

free(records3);

free(records4);

return 1;

}

// Wczytanie rekordow z plikow wejsciowych

printf("\nDane z pliku wejsciowego 1:\n");

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

fread(&records1[i], sizeof(Record), 1, fpIn1);

printf("%d %c ", records1[i].data, records1[i].character);

if ((i + 1) % RECORDS\_PER\_LINE == 0) {

printf("\n");

}

}

printf("\nDane z pliku wejsciowego 2:\n");

for (i = 0; i < n2; i++) {

fread(&records2[i], sizeof(Record), 1, fpIn2);

printf("%d %c ", records2[i].data, records2[i].character);

if ((i + 1) % RECORDS\_PER\_LINE == 0) {

printf("\n");

}

}

printf("\nDane z pliku wejsciowego 3:\n");

for (i = 0; i < n3; i++) {

fread(&records3[i], sizeof(Record), 1, fpIn3);

printf("%d %c ", records3[i].data, records3[i].character);

if ((i + 1) % RECORDS\_PER\_LINE == 0) {

printf("\n");

}

}

printf("\nDane z pliku wejsciowego 4:\n");

for (i = 0; i < n4; i++) {

fread(&records4[i], sizeof(Record), 1, fpIn4);

printf("%d %c ", records4[i].data, records4[i].character);

if ((i + 1) % RECORDS\_PER\_LINE == 0) {

printf("\n");

}

}

// Zamkniecie plikow wejsciowych

fclose(fpIn1);

fclose(fpIn2);

fclose(fpIn3);

fclose(fpIn4);

// Łączenie i sortowanie rekordow

int n5 = n1 + n2 + n3 + n4;

Record \*records5 = (Record \*)malloc(n5 \* sizeof(Record));

if (records5 == NULL) {

printf("Blad alokacji pamieci.\n");

free(records1);

free(records2);

free(records3);

free(records4);

return 1;

}

i = 0;

j = 0;

k = 0;

l = 0;

m = 0;

while (i < n1 && j < n2 && k < n3 && l < n4) {

if (records1[i].data < records2[j].data && records1[i].data < records3[k].data && records1[i].data < records4[l].data) {

records5[m++] = records1[i++];

} else if (records2[j].data < records1[i].data && records2[j].data < records3[k].data && records2[j].data < records4[l].data) {

records5[m++] = records2[j++];

} else if (records3[k].data < records1[i].data && records3[k].data < records2[j].data && records3[k].data < records4[l].data) {

records5[m++] = records3[k++];

} else {

records5[m++] = records4[l++];

}

}

while (i < n1 && j < n2 && k < n3) {

if (records1[i].data < records2[j].data && records1[i].data < records3[k].data) {

records5[m++] = records1[i++];

} else if (records2[j].data < records1[i].data && records2[j].data < records3[k].data) {

records5[m++] = records2[j++];

} else {

records5[m++] = records3[k++];

}

}

while (i < n1 && j < n2 && l < n4) {

if (records1[i].data < records2[j].data && records1[i].data < records4[l].data) {

records5[m++] = records1[i++];

} else if (records2[j].data < records1[i].data && records2[j].data < records4[l].data) {

records5[m++] = records2[j++];

} else {

records5[m++] = records4[l++];

}

}

while (i < n1 && k < n3 && l < n4) {

if (records1[i].data < records3[k].data && records1[i].data < records4[l].data) {

records5[m++] = records1[i++];

} else if (records3[k].data < records1[i].data && records3[k].data < records4[l].data) {

records5[m++] = records3[k++];

} else {

records5[m++] = records4[l++];

}

}

while (j < n2 && k < n3 && l < n4) {

if (records2[j].data < records3[k].data && records2[j].data < records4[l].data) {

records5[m++] = records2[j++];

} else if (records3[k].data < records2[j].data && records3[k].data < records4[l].data) {

records5[m++] = records3[k++];

} else {

records5[m++] = records4[l++];

}

}

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

if (records1[i].data < records2[j].data) {

records5[m++] = records1[i++];

} else {

records5[m++] = records2[j++];

}

}

while (i < n1 && k < n3) {

if (records1[i].data < records3[k].data) {

records5[m++] = records1[i++];

} else {

records5[m++] = records3[k++];

}

}

while (i < n1 && l < n4) {

if (records1[i].data < records4[l].data) {

records5[m++] = records1[i++];

} else {

records5[m++] = records4[l++];

}

}

while (j < n2 && k < n3) {

if (records2[j].data < records3[k].data) {

records5[m++] = records2[j++];

} else {

records5[m++] = records3[k++];

}

}

while (j < n2 && l < n4) {

if (records2[j].data < records4[l].data) {

records5[m++] = records2[j++];

} else {

records5[m++] = records4[l++];

}

}

while (k < n3 && l < n4) {

if (records3[k].data < records4[l].data) {

records5[m++] = records3[k++];

} else {

records5[m++] = records4[l++];

}

}

while (i < n1) {

records5[m++] = records1[i++];

}

while (j < n2) {

records5[m++] = records2[j++];

}

while (k < n3) {

records5[m++] = records3[k++];

}

while (l < n4) {

records5[m++] = records4[l++];

}

// Otwarcie pliku wyjsciowego

FILE \*fpOut = fopen(filenameOut, "wb");

if (fpOut == NULL) {

printf("Blad otwarcia pliku wyjsciowego.\n");

free(records1);

free(records2);

free(records3);

free(records4);

free(records5);

return 1;

}

// Zapisanie posortowanych rekordow do pliku wyjsciowego

printf("\nDane z pliku wyjsciowego:\n");

for (i = 0; i < n5; i++) {

fwrite(&records5[i], sizeof(Record), 1, fpOut);

printf("%d %c ", records5[i].data, records5[i].character);

if ((i + 1) % RECORDS\_PER\_LINE == 0) {

printf("\n");

}

}

// Zamkniecie plikow

fclose(fpOut);

free(records1);

free(records2);

free(records3);

free(records4);

free(records5);

printf("\nRekordy zapisano do pliku %s.\n", filenameOut);

return 0;

}

1. Dane wejściowe

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

1. Dane wyjściowe

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

1. Złożoność obliczeniowa algorytmu

O(1)