**SPRAWOZDANIE**

|  |  |  |  |
| --- | --- | --- | --- |
| **Przedmiot** | Wprowadzenie do Informatyki | **Zadanie** | 9.4 |
| **Autor** | Mateusz Jasiński | **Grupa** | WCY20IY2S1 |
| **Temat** | Pliki i strumienie – pliki binarne. | | |

1. Treść

Program łączy pliki binarne zawierające posortowane wg pola int tak, aby zapisując je do piątego pliku były również posortowane oraz wyświetla na ekranie (dane każdego rekordu w kolejnym wierszu). Listowanie zaczyna nagłówek.

* 1. Metoda realizacji

Odczytujemy z czterech plików rekordy do czterech tablic, następnie dwie tablice scalamy do jednej tablicy, dwie pozostałe tablice również scalamy, na koniec scalamy powstałe tablice do tablicy końcowej i wypisujemy jej elementy na ekran oraz zapisujemy je do piątego pliku.

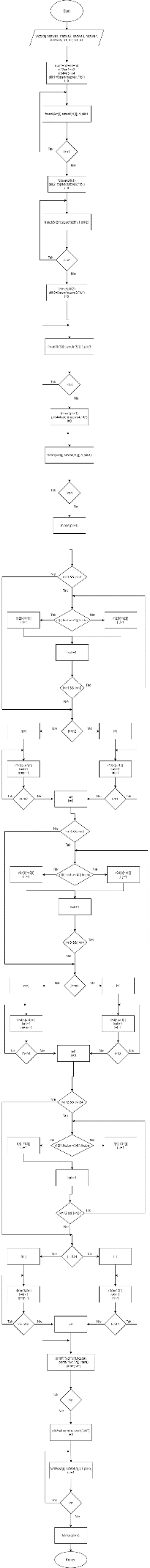
* 1. Założenia / ograniczenia dotyczące danych:
     1. Dane wejściowe

Ilość n1 rekordów z pierwszego pliku, ilość n2 rekordów z drugiego pliku, ilość n3 rekordów z trzeciego pliku, ilość n4 rekordów z czwartego pliku, nazwy pięciu plików – wprowadzane z klawiatury.

* + 1. Dane wyjściowe

n posortowanych rekordów – wyprowadzane na ekran i zapisane do piątego pliku.

1. Realizacja
   1. Algorytm



* 1. Kod źródłowy

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct pola {

int i;

char ch;

} pola;

pola \*help(char \*nazwa\_pliku, pola \*tab, int \*n) {

FILE\* plik=fopen(nazwa\_pliku,"rb");

pola \*temp;

if(!plik) {

printf("Nie znaleziono pliku do wczytania!");

exit(0);

}

int i;

while(fread(&temp, sizeof(tab), 1, plik)!='\0')

\*n=\*n+1;

rewind(plik);

tab=(pola\*)malloc(\*n\*sizeof(tab));

for(i=0;i<\*n;i++)

fread(tab+i, sizeof(tab), 1, plik);

fclose(plik);

return tab;

}

pola f1\_4(int \*x1, int n1, int \*x2, int n2, int \*x3, int n3, int \*x4, int n4, pola \*tab1, pola \*tab2, pola \*tab3, pola \*tab4) {

if(tab1[\*x1].i<tab2[\*x2].i && tab1[\*x1].i<tab3[\*x3].i && tab1[\*x1].i<tab4[\*x4].i) {

\*x1=\*x1+1;

return tab1[\*x1-1];

}

else if(tab2[\*x2].i<tab3[\*x3].i && tab2[\*x2].i<tab4[\*x4].i) {

\*x2=\*x2+1;

return tab2[\*x2-1];

} else if(tab3[\*x3].i<tab4[\*x4].i) {

\*x3=\*x3+1;

return tab3[\*x3-1];

} else {

\*x4=\*x4+1;

return tab4[\*x4-1];

}

}

pola f1\_3(int \*x2, int n2, int \*x3, int n3, int \*x4, int n4, pola \*tab2, pola \*tab3, pola \*tab4) {

if(tab2[\*x2].i<tab3[\*x3].i && tab2[\*x2].i<tab4[\*x4].i) {

\*x2=\*x2+1;

return tab2[\*x2-1];

} else if(tab3[\*x3].i<tab4[\*x4].i) {

\*x3=\*x3+1;

return tab3[\*x3-1];

} else {

\*x4=\*x4+1;

return tab4[\*x4-1];

}

}

pola f1\_2(int \*x3, int n3, int \*x4, int n4, pola \*tab3, pola \*tab4) {

if(tab3[\*x3].i<tab4[\*x4].i) {

\*x3=\*x3+1;

return tab3[\*x3-1];

} else {

\*x4=\*x4+1;

return tab4[\*x4-1];

}

}

pola \*wczytaj(char \*nazwa\_pliku1, char \*nazwa\_pliku2, char \*nazwa\_pliku3, char \*nazwa\_pliku4, pola \*tab, int \*n) {

int i, n1=0, n2=0, n3=0, n4=0, x1, x2, x3, x4;

pola \*tab1, \*tab2, \*tab3, \*tab4;

tab1=help(nazwa\_pliku1, tab1, &n1);

tab2=help(nazwa\_pliku2, tab2, &n2);

tab3=help(nazwa\_pliku3, tab3, &n3);

tab4=help(nazwa\_pliku4, tab4, &n4);

\*n=n1+n2+n3+n4;

tab=(pola\*)malloc(\*n\*sizeof(pola));

for(i=0,x1=0,x2=0,x3=0,x4=0;x1<n1||x2<n2||x3<n3||x4<n4;i++) {

if(x1<n1 && x2<n2 && x3<n3 && x4<n4)

tab[i]=f1\_4(&x1,n1,&x2,n2,&x3,n3,&x4,n4,tab1,tab2,tab3,tab4);

else if(x1<n1 && x2<n2 && x3<n3)

tab[i]=f1\_3(&x1,n1, &x2,n2, &x3,n3, tab1,tab2,tab3);

else if(x1<n1 && x2<n2 && x4<n4)

tab[i]=f1\_3(&x1,n1, &x2,n2, &x4,n4, tab1,tab2,tab4);

else if(x1<n1 && x3<n3 && x4<n4)

tab[i]=f1\_3(&x1,n1, &x4,n4, &x3,n3, tab1,tab4,tab3);

else if(x2<n2 && x3<n3 && x4<n4)

tab[i]=f1\_3(&x4,n4, &x2,n2, &x3,n3, tab4,tab2,tab3);

else if(x1<n1 && x2<n2)

tab[i]=f1\_2(&x1,n1, &x2,n2, tab1,tab2);

else if(x1<n1 && x3<n3)

tab[i]=f1\_2(&x1,n1, &x3,n3, tab1,tab3);

else if(x1<n1 && x4<n4)

tab[i]=f1\_2(&x1,n1, &x4,n4, tab1,tab4);

else if(x2<n2 && x3<n3)

tab[i]=f1\_2(&x2,n2, &x3,n3, tab2,tab3);

else if(x2<n2 && x4<n4)

tab[i]=f1\_2(&x2,n2, &x4,n4, tab2,tab4);

else if(x3<n3 && x4<n4)

tab[i]=f1\_2(&x3,n3, &x4,n4, tab3,tab4);

else if(x1<n1) {

tab[i]=tab1[x1];

x1++;

} else if(x2<n2) {

tab[i]=tab2[x2];

x2++;

} else if(x3<n3) {

tab[i]=tab3[x3];

x3++;

} else if(x4<n4) {

tab[i]=tab4[x4];

x4++;

}

}

free(tab1);

free(tab2);

free(tab3);

free(tab4);

return tab;

}

zapisz(char \*nazwa\_pliku, pola \*tab, int n) {

FILE\* plik=fopen(nazwa\_pliku,"wb");

if(!plik) {

printf("Nie udalo sie utworzyc pliku!");

exit(0);

}

int i;

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

fwrite(tab+i, sizeof(tab), 1, plik);

fclose(plik);

}

main() {

int n, i;

pola \*tab;

char \*nazwa\_pliku1=(char\*)calloc(50, sizeof(char\*));

char \*nazwa\_pliku2=(char\*)calloc(50, sizeof(char\*));

char \*nazwa\_pliku3=(char\*)calloc(50, sizeof(char\*));

char \*nazwa\_pliku4=(char\*)calloc(50, sizeof(char\*));

char \*nazwa\_pliku5=(char\*)calloc(50, sizeof(char\*));

printf("Podaj nazwe pliku1 do wczytania wartosci: ");

gets(nazwa\_pliku1);

printf("Podaj nazwe pliku2 do wczytania wartosci: ");

gets(nazwa\_pliku2);

printf("Podaj nazwe pliku3 do wczytania wartosci: ");

gets(nazwa\_pliku3);

printf("Podaj nazwe pliku4 do wczytania wartosci: ");

gets(nazwa\_pliku4);

printf("Podaj nazwe pliku do zapisania wartosci: ");

gets(nazwa\_pliku5);

tab=wczytaj(nazwa\_pliku1, nazwa\_pliku2, nazwa\_pliku3, nazwa\_pliku4, tab, &n);

printf("\nPosortowane liczby:\nint\tchar\n");

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

printf("%d\t%c\n",tab[i].i,tab[i].ch);

zapisz(nazwa\_pliku5, tab, n);

free(nazwa\_pliku1);

free(nazwa\_pliku2);

free(nazwa\_pliku3);

free(nazwa\_pliku4);

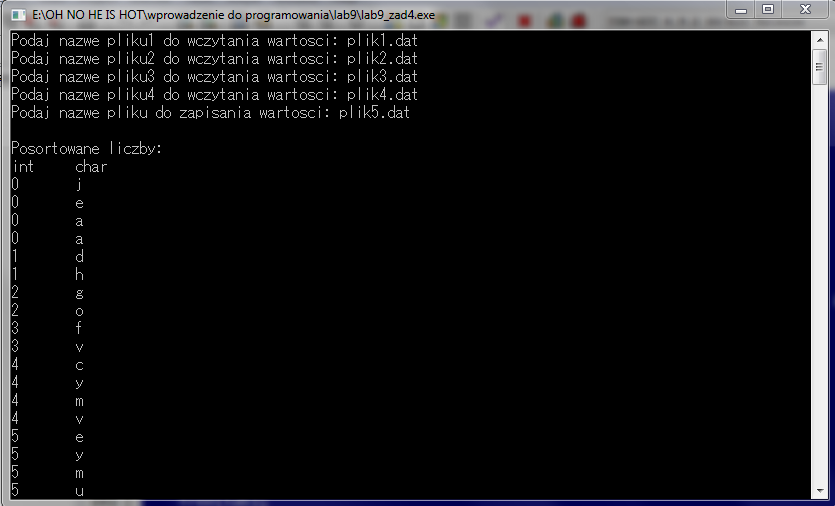
free(nazwa\_pliku5);

free(tab);

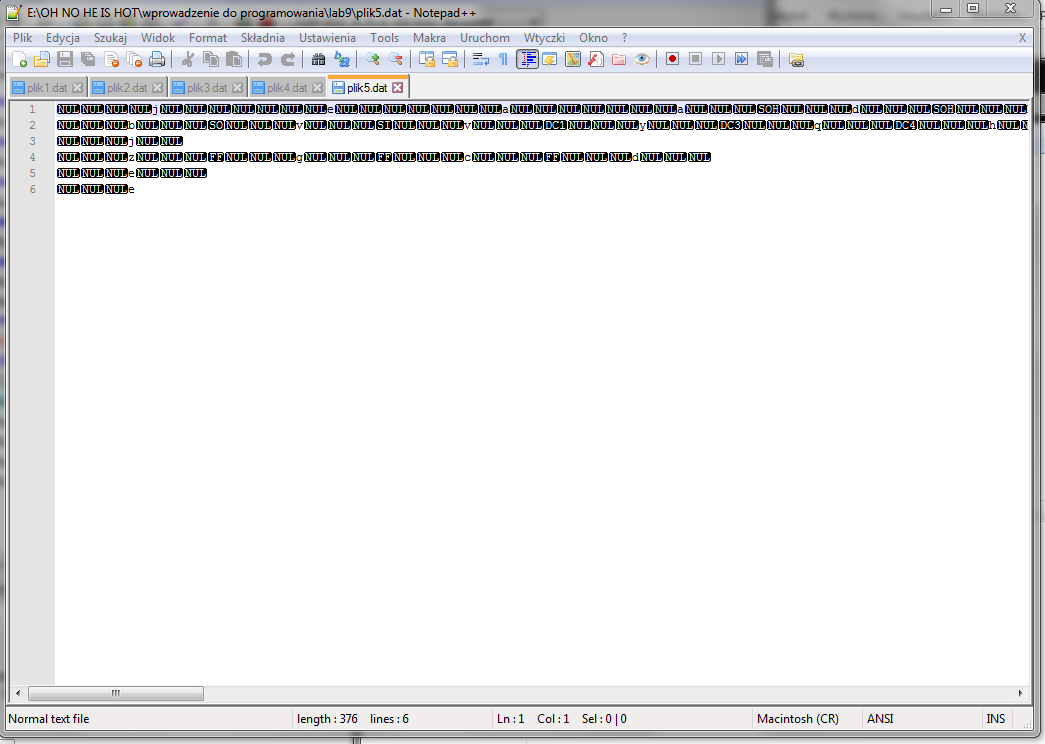
return 0;

}

* 1. Dane wejściowe



* 1. Dane wyjściowe



1. Wnioski

Złożoność obliczeniowa algorytmu:

O(n) = 2 + n1 + 1 + n2 + 1 + n3 + n4 + 1 + n1 + n2 + n3 + n4 + n + n = 2n + 2n + 5 = 4n + 5