Wykład 5_1
Sortowanie zewnętrzne

1. **Algorytm sortowania za pomocą łączenia naturalnego**
2. **Algorytm sortowania za pomocą wielokierunkowego łączenia wyważonego**

1. Algorytm sortowania za pomocą łączenia naturalnego

Przykład 1.

Zawartość pliku złożonego z 20 elementów - 10 serii

Plik	indeksy	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
źródłowy	wartości	-1	-4	0	5	7	4	-4	8	-1	5	9	2	7	4	7	9	-5	-2	-5	-6

1.Podział na 9 serii rozłożonych na dwóch plikach; po złączeniu uzyskano 5 serii

Plik 1	indeksy	0	1	2	3	4	5	6	7	8										
	wartości	-1	4	-1	5	9	4	7	9	-5										
Plik 2	indeksy	0	1	2	3	4	5	6	7	8	9	10								
	wartości	-4	0	5	7	-4	8	2	7	-5	-2	-6								
Plik 0	indeksy	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	wartości	-4	-1	0	4	5	7	-4	-1	5	8	9	2	4	7	7	9	-5	-5	-2	-6

2.Podział na 5 serii rozłożonych na dwóch plikach; po złączeniu uzyskano 3 serie

Plik 1	indeksy	0	1	2	3	4	5	6	7	8	9	10	11							
	wartości	-4	-1	0	4	5	7	2	4	7	7	9	-6							
Plik 2	indeksy	0	1	2	3	4	5	6	7											
	wartości	-4	-1	5	8	9	-5	-5	-2											
Plik 0	indeksy	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	wartości	-4	-4	-1	-1	0	4	5	5	7	8	9	-5	-5	-2	2	4	7	7	9	-6

3. Podział na 3 serie rozłożone na dwóch plikach; po złączeniu uzyskano 2 serie

Plik 1	indeksy	0	1	2	3	4	5	6	7	8	9	10	11							
	wartości	-4	-4	-1	-1	0	4	5	5	7	8	9	-6							
Plik 2	indeksy	0	1	2	3	4	5	6	7											
	wartości	-5	-5	-2	2	4	7	7	9											
Plik 0	indeksy	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	wartości	-5	-5	-4	-4	-2	-1	-1	0	2	4	4	5	5	7	7	7	8	9	9	-6

4. Podział na 2 serie rozłożone na dwóch plikach; po złączeniu uzyskano 1 serię

Plik 1	indeksy	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	wartości	-5	-5	-4	-4	-2	-1	-1	0	2	4	4	5	5	7	7	7	8	9	9
Plik 2	indeksy	0																		
	wartości	-6																		
Plik 0	indeksy	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	wartości	-6	-5	-5	-4	-4	-2	-1	-1	0	2	4	4	5	5	7	7	7	8	9	9

Definicja serii:
Ciąg *a1,...an* taki, że :

$a_k \leq a_{k+1}$ *dla* $k = i...j-1$

$a_{i-1} > a_i$

$a_j > a_{j+1}$

będziemy nazywać największą serią lub krótko serię

Algorytm sortowania za pomocą łączenia naturalnego-poziom konceptualny

(1) *Dopóki nie połączysz elementów z wszystkich serii w jedną serię, wykonuj:*
- ***(1.1)*** *Ustaw plik wejściowy oraz dwa pliki robocze*
- ***(1.2)*** *Podziel plik źródłowy na serie i rozłóż je równomiernie na dwóch plikach roboczych;*
- ***(1.3)*** *Ustaw plik wyjściowy oraz pliki robocze;*
- ***(1.4)*** *Dopóki nie wyczerpiesz serii ze wszystkich plików roboczych, wykonuj:*
  - ***(1.4.1*** *Weź po jednej serii o tym samym numerze z każdego niewyczerpanego pliku roboczego;*
  - ***(1.4.2)*** *Połącz te serie i umieść w pliku wyjściowym w postaci jednej serii;*
  - ***(1.4.3)*** *Wyznacz kolejne serie o tych samych numerach, po jednej z każdego niewyczerpanego pliku roboczego;*

Algorytm sortowania za pomocą łączenia naturalnego- poziom projektowy

l - liczba serii;
plik - file of obiekt; f0: plik - plik źródłowy; f1,f2: plik - pliki robocze
bufor : obiekt - bufor do czytania i zapisu danych w pliku
koniec_serii i : boolean - znacznik końca serii

kopiuj(f_1, f_2):
(1) *Read(f_1, bufor);*
(2) *Write(f_2, bufor);*
(3) *jeśli Eof(f_1), to koniec_serii ← True, w przeciwnym wypadku:*
koniec_serii ← bufor.klucz > f_1↑.klucz;

kopiuj_serie(f_1, f_2):
(1) *dopóki not koniec_serii, wykonuj, co następuje:*
(1.1) *kopiuj(f_1, f_2);*

łącz_serie:
(1) *dopóki koniec_serii <> True, wykonuj co następuje:*
(1.1) *jeśli f1↑.klucz < f2↑.klucz, to:*
(1.1.1) *kopiuj(f1, f0);*
(1.1.2) *jeśli koniec_serii, to kopiuj_serie(f2, f0);*
(1.2) *w przeciwnym wypadku:*
(1.2.1) *kopiuj(f2, f0);*
(1.2.2) *jeśli koniec_serii, to kopiuj_serie(f1, f0);*

Sortowanie_łączenie naturalne(f0, f1, f2)
(1) *dopóki l > 1, wykonuj co następuje:*
(1.1) *Rewrite(f1); Rewrite(f2); Reset(f0)*
(1.2) *dopóki not Eof(f0), wykonuj co następuje:*
(1.2.2) *kopiuj_serie(f0,f1)*
(1.2.3) *jeœli not Eof(f0), to kopiuj_serie(f0,f2)*
(1.3) *Reset(f1); Reset(f2); Rewrite(f0);*
(1.4) *l ← 0;*
(1.5) *dopóki Eof(f1) lub Eof(f2), wykonuj co następuje:*
(1.5.1) *łącz_serie;*
(1.5.2) *l ← l + 1;*
(1.6) *dopóki not Eof(f1), wykonuj, co następuje:*
(1.6.1) *kopiuj_serie(f1, f0);*
(1.6.2) *l ← l + 1;*
(1.7) *dopóki not Eof(f2), wykonuj, co następuje:*
(1.7.1) *kopiuj_serie(f2,f0);*
l ← l + 1;

```
    //program Sortowanie_laczenie_naturalne;
//---------------------------------------------------------------------------
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

const char nazwa[] = "seria_N";
const int _false=0;
const int _true=1;
struct obiekt
 {  int klucz;
 };

unsigned long filesize(FILE *plik);
inline FILE* _reset(FILE* f, const char nazwa[]);
inline FILE* _rewrite(FILE* f, const char nazwa[]);
inline
 void kopiuj(FILE* f0, FILE* f, int& koniec_serii, int& pisz, obiekt&
buf2);
void kopiuj_serie(FILE* f0, FILE* f, obiekt& buf, int& pisz);
void rozdziel_serie_na_dwa_pliki(FILE* f0, FILE* f1, FILE* f2);
void polacz_serie(FILE* f0, FILE* f1, FILE* f2, obiekt& buf1,
                  obiekt& buf2, int& pisz1, int& pisz2);
 void polacz_serie_z_dwoch_plikow(FILE* f0, FILE* f1, FILE* f2,
                                  long& liczba_serii);
void plikowe_sortowanie_laczenie_naturalne(const char nazwa[]);

void wygeneruj_losowo_zawartosc_pliku(const char nazwa[]);
void wydruk(const char nazwa[]);

void main(int argc, char* argv[])
{
  clrscr();
  wygeneruj_losowo_zawartosc_pliku(nazwa);     //generuj losowo plik plik0
  wydruk(nazwa);
  plikowe_sortowanie_laczenie_naturalne(nazwa);
  wydruk(nazwa);
}
```

```
void wygeneruj_losowo_zawartosc_pliku(const char nazwa[])
 {  obiekt buf;                    //element pliku
    long dl; int los;               //dl-rozmiar pliku, los-uzywane przy generacji pliku
    FILE* plik;

    plik=fopen(nazwa,"wb");
    dl=10; los=7789;
    while(dl!=0)
     { los=(131071*los) % 2147483647;
       buf.klucz=los / 21474840;
       fwrite(&buf, sizeof(obiekt),1,plik);
       dl--;
      }
    fclose(plik);
 }

void wydruk(const char nazwa[])
 { long z;  obiekt buf;  FILE* plik;
  plik=fopen (nazwa,"rb");     //jesli nie mozna otworzyc pliku,
  if (plik==NULL) exit(1);       //nalezy przerwac program
  printf("%s\n",nazwa);
  z=0;
  while (fread(&buf, sizeof(buf),1,plik)==1)  //jesli nie osoiagniwto konca pliku
    { printf("%5d",buf.klucz);
       z++;
       if (z % 300 == 0)
          if (getch()=='k') break;}
   printf(" koniec\n");getch();
   fclose(plik);
  }

unsigned long filesize(FILE *plik)
   {  long bpozycja, rozmiar;
       bpozycja = ftell(plik);
       fseek(plik, 0L, SEEK_END);
       rozmiar = ftell(plik);
       fseek(plik, bpozycja, SEEK_SET);
       return rozmiar;}
```

```
 inline FILE* _reset(FILE* f, const char nazwa[])
  { fclose(f);
    return fopen(nazwa,"rb"); }

inline FILE* _rewrite(FILE* f, const char nazwa[])
  { fclose(f);
    return fopen(nazwa,"wb");}

inline void kopiuj(FILE* f0, FILE* f,
                   int& koniec_serii, int& pisz, obiekt& buf2)
  { obiekt buf1;  int _eof;
    if (ftell(f0)==0)
       { fread(&buf1, sizeof(buf1),1,f0);
         fwrite(&buf1, sizeof(buf1),1,f); }
    else
       { fwrite(&buf2, sizeof(buf2),1,f);
         buf1=buf2;}

     if ( fread(&buf2,sizeof(buf2), 1, f0)==1)   pisz=_true;
     else                                        pisz=_false;
     if (!pisz)        koniec_serii=_true;
     else              koniec_serii= buf1.klucz > buf2.klucz;
 }

void kopiuj_serie(FILE* f0, FILE* f, obiekt& buf, int& pisz)
  { int koniec_serii;
    do
     { kopiuj(f0,f,koniec_serii,pisz,buf);
     } while(!koniec_serii);
  }

void rozdziel_serie_na_dwa_pliki(FILE* f0, FILE* f1, FILE* f2)
  { obiekt buf;     int pisz;
    do
       { kopiuj_serie(f0,f1,buf,pisz);
         if (pisz) kopiuj_serie(f0,f2,buf,pisz);
       } while (pisz);
   }
```

```
void polacz_serie(FILE* f0, FILE* f1, FILE* f2,
                  obiekt& buf1, obiekt& buf2, int& pisz1, int& pisz2)
 { int koniec_serii;

   do
    { if (buf1.klucz < buf2.klucz)
        { kopiuj(f1,f0,koniec_serii,pisz1,buf1);
          if (koniec_serii) kopiuj_serie(f2,f0,buf2,pisz2);}
      else
        { kopiuj(f2,f0,koniec_serii,pisz2,buf2);
          if (koniec_serii) kopiuj_serie(f1,f0,buf1,pisz1);}
    } while(!koniec_serii);
 }

void polacz_serie_z_dwoch_plikow(FILE* f0, FILE* f1, FILE* f2,
                                 long& liczba_serii)
 { obiekt buf1,buf2;
   int  pisz1,pisz2;

   if (fread(&buf1,sizeof(buf1),1,f1)!=1) return;
   if (fread(&buf2,sizeof(buf2),1,f2)!=1) return;
   f0=_rewrite(f0,nazwa);
   do
    { polacz_serie(f0, f1, f2, buf1, buf2, pisz1, pisz2);
      liczba_serii++;
    } while (pisz1 && pisz2);
  while (pisz1)
    { kopiuj_serie(f1,f0,buf1,pisz1);   liczba_serii++;}
  while (pisz2)
    { kopiuj_serie(f2,f0,buf2,pisz2);   liczba_serii++;}
 }
```

```
void plikowe_sortowanie_laczenie_naturalne(const char nazwa[])
 {
   long liczba_serii;                          //liczba rozłożonych serii
    FILE *f0,*f1,*f2;                          //fo -plik główny, f1 i f2-pliki pomocnicze
    const char nazwa1[]="seria1_N";            //nazwy plików pomocniczych
    const char nazwa2[]="seria2_N";

  f1=fopen(nazwa1,"wb");
  f2=fopen(nazwa2,"wb");
  f0=fopen(nazwa,"rb");
  if (filesize(f0) >= 2*sizeof(obiekt))
    do
      {  f1=_rewrite(f1,nazwa1);   //przygotowanie plików do rozdzielania serii
         f2=_rewrite(f2,nazwa2);    //f1, f2 – wyjścia, fo - wejście
         f0=_reset(f0,nazwa);
         rozdziel_serie_na_dwa_pliki(f0,f1,f2);

         f1=_reset(f1,nazwa1);      //przygotowanie plików do łączenia serii
         f2=_reset(f2,nazwa2);      //f1, f2 – wejścia, fo-wyjście
         liczba_serii=0;
         polacz_serie_z_dwoch_plikow(f0, f1, f2,liczba_serii);
    } while (liczba_serii>1);

  fclose(f0);                                  //posortowany plik
  fclose(f1);   remove(nazwa1);   //usuwanie plików pomocniczych
  fclose(f2);   remove(nazwa2);
 }   //koniec plikowe_sortowanie_laczenie_natualne
```

2. Algorytm sortowania za pomocą wielokierunkowego łączenia wyważonego

Przykład 2

Zawartość pliku złożonego z 20 elementów - 10 serii

Plik	indeksy	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
źródłowy	wartości	-1	-4	0	5	7	4	-4	8	-1	5	9	2	7	4	7	9	-5	-2	-5	-6

1. Podział na 10 serii rozłożonych równomiernie na trzech plikach

Plik 1	indeksy	0	1	2	3	4	5	6
	wartości	-1	-4	8	4	7	9	-6

Plik 2	indeksy	0	1	2	3	4	5	6	7	8
	wartości	-4	0	5	7	-1	5	9	-5	-2

Plik 3	indeksy	0	1	2	3
	wartości	4	2	7	-5

2. Łączenie i podział serii za pomocą 6 plików

2.1. Pierwsza faza - uzyskanie 4 serii na 3 plikach

Plik4	indeksy	0	1	2	3	4	5	6
	wartości	-4	-1	0	4	5	7	-6

Plik 5	indeksy	0	1	2	3	4	5	6
	wartości	-4	-1	2	5	7	8	9

Plik 6	indeksy	0	1	2	3	4	5
	wartości	-5	-5	-2	4	7	9

2.2. Druga faza - uzyskanie 2 serii na 2 plikach

Plik 1	indeksy	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	wartości	-5	-5	-4	-4	-2	-1	-1	0	2	4	4	5	5	7	7	7	8	9	9

Plik 2	indeksy	0
	wartości	-6

2.3. Trzecia faza - uzyskanie jednej serii na jednym pliku- koniec sortowania

Plik 4	indeksy	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	wartości	-6	-5	-5	-4	-4	-2	-1	-1	0	2	4	4	5	5	7	7	7	8	9	9

Nr pliku	Uwagi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1. plik 4	plik 2	-4																			
	plik 1-kc		-1																		
	plik 2			0																	
	plik 3-kc				4																
	plik 2-kc					5	7														
plik 5	plik 1	-4																			
	plik 2		-1																		
	plik 3			2																	
	plik 2				5																
	plik 3-kc					7															
	plik 1-kc						8														
	plik 2-kc							9													
plik 6	plik 2	-5																			
	plik 3-kx		-5																		
	plik 2-kx			-2																	
	plik 1-kc				4	7	9														
plik 4	plik 1-kx							-6													
2. plik 1	plik 6	-5																			
	plik 6		-5																		
	plik 4			-4																	
	plik 5				-4																
	plik 6					-2															
	plik 4						-1														
	plik 5							-1													
	plik 4								0												
	plik 5									2											
	plik 4										4										
	plik 6											4									
	plik 4												5								
	plik 5													5							
	plik 4-kc														7						
	plik 6															7					
	plik 5																7				
	plik 5																	8			
	plik 6-kx																		9		
	plik 5-kx																			9	
plik 2	plik 4-kx	-6																			
3. plik 4	plik 2-kx	-6																			
	plik 1		-5	-5	-4	-4	-2	-1	-1	0	2	4	4	5	5	7	7	7	8	9	9

Algorytm sortowania za pomocą wielokierunkowego łączenia wyważonego - poziom konceptualny

(1) *Podziel plik źródłowy na serie i rozłóż je równomiernie na połowie plików roboczych;*

(2) *Dopóki nie połączysz elementów z wszystkich serii w jedną serię, wykonuj:*

- ***(2.1)*** *Ustaw pliki wejściowe; {pliki robocze z rozdzielonymi seriami w (1)}*
- ***(2.2)*** *Ustaw pliki wyjściowe; {druga polowa plików roboczych}*
- ***(2.2)*** *Dopóki nie wyczerpiesz serii ze wszystkich plików wejściowych, wykonuj:*
  - ***(2.2.1)*** *Weź po jednej serii o tym samym numerze z każdego niewyczerpanego pliku wejściowego;*
  - ***(2.2.3)*** *Wyznacz kolejny plik wyjściowy, a jeżeli zostały wyczerpane, weź pierwszy plik wyjściowy;*
  - ***(2.2.2)*** *Połącz wyznaczone serie i umieść na wybranym pliku wyjściowym w postaci jednej serii;*
  - ***(2.2.4)*** *Wyznacz kolejne serie, po jednej o tym samym numerze z każdego niewyczerpanego pliku wejściowego;*
- ***(2.3)*** *Zamień pliki wyjściowe na wejściowe;*

Algorytm wielokierunkowego łączenia wyważonego - poziom projektowy

n - liczba plikow; l - liczba serii;
k1 - liczba plikow wejściowych; k2 - liczba aktywnych plików wejściowych
i-numery na mapie plików wejściowych; j-numery na mapie plików wyjściowych
plik - file of obiekt; f0: plik - plik źródłowy;
pliki= array[nrpliku] of plik; f: pliki- tablica plików wejściowych i wyjściowych
bufor : obiekt - bufor do czytania i zapisu danych w pliku
mapa_plikow= array[nrpliku] of nrpliku;
t, ta : mapa_plikow - mapy numerów plikow wejściowych i wyjściowych

(1) *j ← nh; l ← 0;*
(2) *dopóki not Eof(f0), wykonuj, co następuje:*
  (2.1) *jeśli j < nh, to j ← j+1, w przeciwnym razie j ← 1;*
  (2.2) *dopóki bufor.klucz > f0↑.klucz ∨ Eof(f0), wykonuj co następuje:*
    (2.2.1) *Read(f0, bufor);*
    (2.2.2) *Write(f[j], bufor);*
  (2.3) *l ← l + 1;*
(3) *i ← 1; wykonaj, co następuje, N razy:*
  (3.1) *t[i] ← i*
  (3.2) *i ← i + 1;*
(4) *dopóki l > 1 wykonuj co następuje:*
  (4.1) *jeśli l < nh, to k1 ← l, w przeciwnym wypadku k1 ← nh*
  (4.2) *i ← 1; wykonaj, co następuje, k1 razy:*
    (4.2.1) *Reset(f[t[i]]);*
    (4.2.2) *ta[i] ← t[i];*
    (4.2.3) *i ← i + 1;*
  (4.3) *l ← 0; j← nh+1; {kolejna faza dzielenia i łączenia serii}*
  (4.4) *dopóki k1 > 0 wykonuj, co następuje:*
    (4.4.1) *l ← l+1;*
    (4.4.2) *k2 ← k1; dopóki k2 > 0 wykonuj, co następuje:*
      (4.4.2.1) *Wybierz najmniejszy klucz i niech ta[mx] będzie numerem jego pliku*
      (4.4.2.2) *Read(f[ta[mx], bufor);*
      (4.4.2.3) *Write(f[t[j], bufor);*
      (4.4.2.4) *jeśli Eof(f[ta[mx]]), to Usun plik {k2 ← k2-1 k1 ← k1-1}, w przeciwnym wypadku: jeśli bufor klucz>f([ta[mx]])↑.klucz, to Zamknij serię {k2 ← k2-1};*
    (4.4.3) *jeśli j < n, to j ← j + 1, w przeciwnym wypadku j ← nh +1;*
  (4.5) *Zamień taśmy wyjściowe na wejściowe.*

```
//sortowanie n-kierunkowe przez laczenie wywazone
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#pragma hdrstop

const int n=6;
const nh=n / 2;                  //liczba plikow
const char nazwa[]="seria";
struct obiekt
 { int klucz;
 };
 struct plik
 { int numer;
   char nazwa[20];
 };
  inline FILE* _reset(FILE* f, const char nazwa[]);
  inline FILE* _rewrite(FILE* f, const char nazwa[]);
  int rozdziel_serie(FILE* f0, FILE* pliki[], long& liczba_serii);
  void uporzadkuj_tab_plikow(plik mapa_plikow[]);
  void polacz_serie(FILE* pliki[],plik mapa_plikow[],int& k1, long& l,
                         obiekt bufory[]);
 void polacz_pliki_wielokierunkowo(long& l, FILE* pliki[],
                                          plik mapa_plikow[]);
 void wielokierunkowe_laczenie(const char nazwa[]);

 void wygeneruj_losowo_zawartosc_pliku(const char nazwa[]);
 void wydruk(const char nazwa[]);

void main(int argc, char* argv[])
{
  clrscr();
  wygeneruj_losowo_zawartosc_pliku(nazwa);     //generuj losowo plik plik0
  wydruk(nazwa);
  wielokierunkowe_laczenie(nazwa);
  wydruk(nazwa);
}
```

```
void wygeneruj_losowo_zawartosc_pliku(const char nazwa[])
 {  obiekt buf;                    //element pliku
    long dl; int los;                 //dl-rozmiar pliku, los-uzywane przy generacji pliku
    FILE* plik;

    plik=fopen(nazwa,"wb");
    dl=50; los=7789;
    while(dl!=0)
     { los=(131071*los) % 2147483647;
       buf.klucz=los / 21474840;
       fwrite(&buf, sizeof(obiekt),1,plik);
       dl--;}
    fclose(plik);
 }

void wydruk(const char nazwa[])
 {long z;
  obiekt buf;
  FILE* plik;

  plik=fopen (nazwa,"rb");    //jesli nie mozna otworzyc pliku,
  if (plik==NULL) exit(1);      //nalezy przerwac program
  printf("%s\n",nazwa);
  z=0;
  while (fread(&buf, sizeof(buf),1,plik)==1)  //jesli nie osiagnieto konca pliku
   { printf("%5d",buf.klucz);
     z++;
     if (z % 300 == 0)
       if (getch()=='k') break;}
   printf(" koniec\n"); getch();
   fclose(plik);
  }

 inline FILE* _reset(FILE* f, const char nazwa[])
  { fclose(f);
    return fopen(nazwa,"rb");
  }

  inline FILE* _rewrite(FILE* f, const char nazwa[])
  { fclose(f);
    return fopen(nazwa,"wb");
  }
```

```
void wielokierunkowe_laczenie(const char nazwa[])
 { int i;
   long serie;                          //liczba rozlozonych serii
   FILE* pliki [n];
   plik mapa_plikow [n];
   FILE* f0;                            //f0 jest plikiem wejsciowym z liczbami losowymi

   f0=fopen(nazwa,"rb");
   if (f0==NULL)  return;
   char z[2];  z[1]='\0';               //przygotuj pliki pomocnicze
   for (int i=0; i< n; i++)
     { z[0]=i+48;
       strcpy(mapa_plikow[i].nazwa,"seria");
       strcat(mapa_plikow[i].nazwa,z);
       pliki[i]=fopen(mapa_plikow[i].nazwa,"wb");}
                          //rozloz poczatkowe serie na mapa_plikow[0]...mapa_plikow[nh-1]
   if (rozdziel_serie(f0,pliki,serie)==0) return;
     polacz_pliki_wielokierunkowo(serie,pliki,mapa_plikow);
   for (int i=0; i< n;i++) fclose(pliki[i]);          //zamknij pliki pomocnicze
       //usun plik zrodlowy nieposortowany i zamien na plik posortowanyzpliki[mapa_plikow[0]]
   fclose(f0); remove(nazwa);
   rename(mapa_plikow[0].nazwa,nazwa);
  }      //koniec plikowe_sortowanie_laczenie}

int rozdziel_serie(FILE* f0, FILE* pliki[], long& liczba_serii)
 {int j=0, koniec_serii =1;
  obiekt buf,buf0;
  liczba_serii=1;
  if (fread(&buf, sizeof(buf), 1, f0)!=1) return 0;
  fwrite(&buf, sizeof(obiekt), 1, pliki[j]);
  while (fread(&buf0,sizeof(obiekt),1,f0)==1)
   { koniec_serii= buf.klucz>buf0.klucz;
     if (koniec_serii)
       { if (j < nh-1) j++; else j=0;
         liczba_serii++;}
     fwrite(&buf0, sizeof(obiekt),1,pliki[j]);
     buf=buf0;}
  return 1;
 }
```

```
void polacz_serie(FILE* pliki[],plik mapa_plikow[],int& k1, long& serie,
                  obiekt bufory[])
 { int i, j, mx, k2,min; / /j=indeks pliku wyjsciowego, k2=liczba aktywnych plikow wejsciowych
   plik ta[n], tx; //pomocnicza mapa plikow ta do elminowanie plikow wyczerpanych
   obiekt buf,buf0;
   for (i=0; i< k1; i++)
     ta[i]=mapa_plikow[i];    //utworz pomocnicza tablice aktywnych plikow laczonych}
   serie=0;                  //l=liczba polaczonych serii
   j=nh;
   //Polacz serie plikow wejsciowych mapa_plikow[0]..mapa_plikow[k+-1] na mapa_plikow[j]}
   while (k1 >0)
    { k2=k1;
      serie++;
      while( k2 > 0)    //Wybierz obiekt najmniejszy w aktywnych plikach wejsciowych}
       {  i=1;     mx=0;
          min=bufory[0].klucz;
          while (i<k2)
           { if (bufory[i].klucz < min)    //wybor pliku ta[mx] z el. min}
              { min=bufory[i].klucz; mx=i;}
              i++; }          //plik ta[mx] ma obiekt najmniejszy, przesun go na
      buf=bufory[mx];         // [plikmapa_plikow[j]
      fwrite(&buf,sizeof(obiekt),1,pliki[mapa_plikow[j].numer]);
      if (fread(&bufory[mx], sizeof(obiekt),1,pliki[ta[mx].numer])!=1)
        { //usun plik po jego wyczerpaniu
          _rewrite(pliki[ta[mx].numer],ta[mx].nazwa);
          bufory[mx]=bufory[k2-1];     bufory[k2-1]=bufory[k1-1];
                  //i uaktualnij tablice aktywnych/ plikow
          ta[mx]=ta[k2-1];             ta[k2-1]=ta[k1-1];
          k1--; k2--;}   //oraz zmniejsz liczbe serii aktywnych k2 oraz wszystkich serii  k1}
       else                                  //zamien pliki, jezeli koniec serii w pliku ta[mx]}
        { if (buf.klucz>bufory[mx].klucz)
            { buf0=bufory[mx];                tx=ta[mx];
              bufory[mx]=bufory[k2-1];        ta[mx]=ta[k2-1];
              bufory[k2-1]=buf0;              ta[k2-1]=tx;
              k2--;  }
         }//koniec kolejnej fazy szukania najmniejszego elementu w aktywnych plikach
     } laczonych  wyznaczenie kolejnego pliku wy. Z mapa_plikow[j]
      if (j<n-1) j++; else j=nh;
    } /*wyczerpane sa wszystkie pliki wejsciowe*/ }
```

```
void uporzadkuj_tab_plikow(plik mapa_plikow[])
 { plik tx;
   for( int i=0; i< nh; i++ )
     { tx=mapa_plikow[i];
       mapa_plikow[i]=mapa_plikow[i+nh];
       mapa_plikow[i+nh]=tx;}
  }

void polacz_pliki_wielokierunkowo(long& serie, FILE* pliki[],
                                  plik mapa_plikow[])
 {  int k1;
    obiekt bufory[n];

   for (int i=0; i< n; i++)
       mapa_plikow[i].numer=i;
         //Lacz z mapa_plikow[0]...mapa_plikow[nh-1] na mapa_plikow[nh]...mapa_plikow[n-1]
         //az do uzyskania jednej serii l=1 w jednym pliku
   while (serie > 1)
    { if (serie<nh) k1=serie;      //Ustaw pliki wejsciowe - ich liczbe k1 oraz otworz je
      else k1=nh;                  //k1=liczba plikow wejsciowych w tej fazie}
      for (int i=0; i<k1; i++)     //pierwszy element z kazdego z plikow laczonych
        { _reset(pliki[mapa_plikow[i].numer],mapa_plikow[i].nazwa);
          fread(&bufory[i],sizeof(obiekt),1,pliki[mapa_plikow[i].numer]); }
      polacz_serie(pliki,mapa_plikow,k1,serie,bufory);
                                   //zamien pliki wyjsciowe na wejsciowe dla kolejnej fazy laczenia
      uporzadkuj_tab_plikow(mapa_plikow);
    }                              //koniec laczenia-koniec sortowania
 }
```