Dr inż. Robert Wójcik

Zakład Podstaw Informatyki i Teleinformatyki
Instytut Cybernetyki Technicznej, Politechnika Wrocławska

Wykład 15

15.1. Pliki tekstowe i binarne

Plik (*plik fizyczny*) - stanowi zbiór danych przechowywanych w pamięci zewnętrznej (np. na twardych dyskach, pamięciach typu flash, taśmach magnetycznych, płytach CD/DVD, itp.), umożliwiający trwałe przechowywanie informacji.

Plik fizyczny – stanowi wydzielony obszar pamięci zewnętrznej i jest identyfikowany przez system operacyjny za pomocą unikalnej nazwy.

Plikami fizycznymi są:

- *pliki dyskowe* – umożliwiające trwałe przechowywanie danych (np. można zapisać dane do pliku i odczytać je ponownie podczas kolejnego uruchomienia programu) lub implementację pamięci wirtualnej, która umożliwia rozszerzenie pamięci operacyjnej poprzez chwilowe przechowywanie jej zawartości na dysku; pliki dyskowe są identyfikowane w systemie operacyjnym za pomocą unikalnej ścieżki dostępu (lista katalogów) i nazwy pliku, np. ”C:\\tmp\\lab\\p.dat”;

- *urządzenia wejścia i wyjścia* (np. klawiatura, monitor, drukarka), które są traktowane przez system operacyjny tak jak pliki fizyczne, mimo iż nie posiadają własności trwałego przechowywania danych.

W języku C każdy plik fizyczny jest ciągiem bajtów, z których każdy może być niezależnie odczytany.

Plik tekstowy – zawiera dane zapisane w postaci znaków czytelnych dla człowieka. Pliki tekstowe mogą być różnie interpretowane w różnych systemach operacyjnych. Wynika to z różnego sposobu interpretacji znaku końca wiersza.

W pliku tekstowym każdy bajt jest kodem ASCII znaku należącym do przedziału od 0 do 255. Na przykład zawartość pliku fizycznego, który zawiera 14 następujących bajtów (hex):

```
42 6F 72 6C 61 6E 64 0D 0A 48 49 50 51 52
```

zostanie w środowisku Dos i Windows zinterpretowana jako tekst:

```
Borland
01234
```

Ciąg znaków OD OA oznacza przejście na początek nowej linii.

Plik binarny – zawiera dane zapisane w postaci binarnej, na ogół trudno czytelnej dla człowieka, np. wersja wykonywalna programu plik.exe.

Dane binarne wykorzystywane w programach komputerowych mogą być zapisywane do plików w postaci:

- binarnej, czyli tak jak są przechowywane w pamięci komputera;

- tekstowej; dane binarne są automatycznie konwertowane na postać znakową i zapisywane do plików tekstowych.

Dane zapisane w plikach mogą być interpretowane przez programy jako dane binarne lub tekstowe. Dane binarne są wczytywane do programów bezpośrednio, natomiast dane interpretowane jako tekstowe są automatycznie konwertowane na postać binarną.

15.2. Standardowa biblioteka wejścia i wyjścia

Obsługa wejścia i wyjścia w języku C wymaga użycia standardowych bibliotek. W języku C standardowe wejście i wyjście wysokiego poziomu wykorzystuje funkcje i definicje zawarte w bibliotece *stdio.h*.

Bibliotekę dołącza się do programu za pomocą deklaracji

```
#include<stdio.h>
```

Biblioteka zawiera funkcje umożliwiające komunikację z plikami oraz standardowymi urządzeniami.

Operacje wejścia (czytanie z plików) i wyjścia (zapisywanie do plików) są realizowane w programach za pośrednictwem odpowiednich struktur danych, nazywanych **strumieniami** (stream), które są reprezentowane przez zmienne plikowe (struktury) typu FILE. Strumienie reprezentują wirtualny przepływ danych pomiędzy programem a plikiem fizycznym i odwrotnie.

Podczas otwierania pliku o podanej nazwie struktura taka jest tworzona automatycznie jako zmienna dynamiczna. Wykonywanie operacji na pliku fizycznym wymaga podania wskaźnika na tę strukturę.

```
FILE *plik1, *plik2;      // zmienne plikowe reprezentujące strumienie
```

Zmienna plikowa określa sposób interpretacji zawartości pliku fizycznego i jest argumentem instrukcji wykonujących operacje na pliku fizycznym. Jest ona wykorzystywana przez system operacyjny do komunikacji między programem a plikiem fizycznym. Przed wykorzystaniem zmienna plikowa musi być skojarzona z odpowiednim plikiem fizycznym lub urządzeniem wejścia / wyjścia.

Standardowe wejście i wyjście jest buforowane. Oznacza to, że dane z programu są przesyłane do tymczasowego bufora w pamięci, np. o rozmiarze 512 bajtów, w którym mogą być przetwarzane, np. na postać znakową, a następnie w całości przesyłane do pliku fizycznego. Analogicznie, podczas odczytu danych z pliku są one wczytywane do tymczasowego bufora, w którym mogą być przetwarzane bajt po bajcie, a następnie przekazywane do zmiennych programu.

W odniesieniu do plików wprowadza się pojęcie wskaźnika pliku określającego pozycję w pliku, z której dana jest odczytywana lub, na której jest zapisywana. W chwili otwarcia pliku bieżącą pozycją w pliku jest pozycja zerowa. Po każdym odczycie elementu z pliku lub zapisie elementu do pliku wskaźnik pliku zostaje przesunięty za odczytany lub zapisany element, tj. jego wskaźnik wzrasta o 1. Odczyt z pliku nie niszczy elementu, natomiast zapis powoduje modyfikację elementu znajdującego się na pozycji wskaźnika pliku.

15.3. Strumienie standardowe

W języku C można wykonywać operacje wejścia i wyjścia z wykorzystaniem strumieni zrealizowanych w postaci strukturalnej (struktury typu FILE). W języku C++ są dostępne również strumienie zrealizowane w postaci obiektowej (obiekty odpowiedniej klasy).

W pliku stdio.h zdefiniowano następujące wskaźniki plikowe skojarzone ze standardowymi strumieniami automatycznie otwieranymi przez programy w języku C/C++ po dołączeniu pliku stdio.h.

Wersja strukturalna (C/C++):
- **stdout** (wskaźnik do struktury) - strumień wyjścia (zwykle ekran; istnieje możliwość przedefiniowania na plik);
- **stdin** (wskaźnik do struktury) - strumień wejścia (zwykle klawiatura; istnieje możliwość przedefiniowania na plik);
- **stderr** (wskaźnik do struktury) - standardowy strumień wyprowadzania błędów (zwykle ekran; istnieje możliwość przedefiniowania na plik).

Wersja obiektowa (C++):
- **cout** (obiekt) - strumień wyjściowy;
- **cin** (obiekt) - strumień wejściowy;
- **cerr** (obiekt) – strumień obsługi błędów.

15.4. Otwieranie plików, tryby otwarcia, zamykanie plików

Dostęp do pliku wymaga otworzenia pliku (strumienia), tj. skojarzenia odpowiedniej zmiennej plikowej z plikiem fizycznym.

- Otwarcie pliku: funkcja fopen zwracająca wskaźnik na FILE.

```
FILE * fopen ( char *nazwa_pliku, char *rodzaj_otwarcia )
```

rodzaj otwarcia:
r – tylko do odczytu
w – tylko do zapisu (utworzenie nowego pliku)
a – dopisywanie na końcu pliku
+ – z możliwością aktualizacji (odczyt i zapis)
t – otwarcie w trybie tekstowym
b – otwarcie w trybie binarnym

Standard ANSI wyróżnia dwa sposoby (tryby) otwierania plików (oba dotyczą plików tekstowych i binarnych).

- Tryb otwarcia tekstowego (t)

- Tryb otwarcia binarnego (b)

Tryby te wynikają z odmiennych sposobów reprezentacji końca wiersza w plikach tekstowych w różnych systemach operacyjnych, a reprezentacją stosowaną w języku C, w którym znak końca jest oznaczany za pomocą znaku \n. W systemie UNIX oba tryby otwarcia są identyczne, gdyż pliki tekstowe i binarne są postrzegane tak jak w języku C.

W systemach DOS oraz Windows koniec wiersza pliku tekstowego jest oznaczany za pomocą sekwencji znaków \r\n, z kolei koniec wiersza w języku C jest reprezentowany przez \znak n. Stąd, jeśli program w języku C zapisuje plik w trybie otwarcia tekstowego (t), to przetwarza bajt \n na sekwencję \r\n. W przypadku odczytu z pliku realizowana jest konwersja odwrotna: każda sekwencja bajtów \r\n jest przekształcana na \n. Konwersja taka jest dokonywana zarówno przy zapisie do plików tekstowych jak i binarnych.

Tryb otwarcia binarnego nie wprowadza modyfikacji danych podczas przesyłania danych pomiędzy plikami a pamięcią.

W praktyce tryb tekstowy stosuje się do plików tekstowych, a tryb binarny do plików binarnych. Można jednak zastosować dowolny tryb otwarcia do dowolnego rodzaju pliku. Należy jednak pamiętać o dokonywanych modyfikacja w trybie tekstowym. W związku z tym należy odczytywać dane w takim trybie w jakim zostały zapisane.

Przykład. 1. Otwieranie pliku

```
FILE   *plik;
   // utworzenie pliku w trybie binarnym z możliwością czytania

       plik = fopen( "dane.dat", "w+b" );
       if ( plik == NULL )              // kontrola błędów we/wy
              {
                     printf( "Blad" );
                     exit( 1 );
              }
```

Zamykanie pliku

- Zamknięcie pliku: funkcja fclose

```
int  fclose ( FILE  *strumien )        // zamknięcie wskazanego strumienia
```

15.5. Zapis danych do plików tekstowych i binarnych

Dane w postaci znakowej można zapisywać do plików tekstowych za pomocą następujących funkcji:

Zapis pojedynczego znaku do strumienia. Jeśli zapis zakończony powodzeniem, to zwracany kod znaku, jeśli nie to zwracany kod błędu.

- `int  fputc ( int znak, FILE  *strumien ) // wysłanie pojedynczego znaku`

Wysłanie łańcucha znaków do strumienia. Nie dołącza znaku końca tekstu \0 oraz końca linii do pliku.

- `int  fputs ( char *tekst, FILE *strumien )       // wysłanie łańcucha znaków`

Funkcja sformatowanego wyjścia analogiczna do printf( ), zapisująca do strumienia.

- `int  fprintf ( FILE *strumien, char *format, arg, ..., arg )`

```
// postać formatu
//  %[znaczniki] [szerokość][.dokładność] [F | N | h | l | L ] <znak typu>
```

Jeśli podamy jako strumień wyjściowy stdout (standardowy strumień wyjściowy), to wtedy dane zostaną wyprowadzone na ekran.

np. `fprintf( stdout,  "format" , ... ) ==  printf( "format" , ... )`

Funkcja zapisu danych w postaci binarnej (pliki binarne):

```
int  fwrite (  void *adres_danej,
               size_t  rozmiar_danej,  size_t  ilosc_danych,
               FILE *strumien)

// funkcja kopiująca (ilosc_danych * rozmiar_danej) bajtów
// spod wskazanego obszaru pamięci do strumienia (pliku)
// zwraca liczbę zapisanych danych (nie bajtów) lub kod błędu
```

Przykład 2. Zapis tablicy struktur do pliku tekstowego i binarnego

```
#include <stdio.h>

const int N=7;
struct  tosoba  {  char  naz[40];   char  data[20];   int  id;  };

void losuj(struct tosoba *s)
{
       sprintf(s->naz,  "naz%d%c", rand() % 99 + 1,0);   //  \0 zero na końcu
       sprintf(s->data, "data%d%c", rand() % 100 + 1900,0);
       s->id = rand() % 99 + 1;
}

void pisz(struct tosoba *s)
{
       printf("%20s ", s->naz);
       printf("%10s ", s->data);
       printf("%8d \n", s->id);
}

void main( void )
{
       FILE *plik;
       struct tosoba mbaza[N];

       for (int i=0; i<N; i++) { losuj(&mbaza[i]); pisz(&mbaza[i]); }

       if ( (plik = fopen( "dane.bin" ,  "wb" ) ) != NULL )
        {
              // zapis zawartości tablicy struktur do pliku binarnego

       fwrite( mbaza, sizeof(struct tosoba1), N , plik);
       // lub  // fwrite( mbaza, sizeof(mbaza), 1 , plik);

       fclose( plik );
              }
```

```
        if ( (plik = fopen("dane.txt" ,  "wt") ) != NULL )
        {
                // zapis zawartości tablicy struktur do pliku tekstowego

        for( int i=0;  i < N;  i++ )
        fprintf ( plik,"%s  %s  %d\n", mbaza[ i ].naz, mbaza[ i ].data,
                                        mbaza[ i ].id );

        fclose( plik );
                }
}
```

15.6. Odczyt danych z plików tekstowych i binarnych

Funkcje odczytu danych w postaci tekstowej (pliki tekstowe):

Odczyt pojedynczego znaku ze strumienia. Jeśli odczyt zakończony powodzeniem, to zwracany kod znaku, jeśli nie to zwracany jest kod błędu.

- int fgetc (FILE *strumien) // wczytanie pojedynczego znaku

Odczyt łańcucha ze strumienia. Czytanie spacji. Koniec czytania – odczyt (dlugosc-1) znaków, wystąpienie znaku końca linii lub końca pliku. Znak końca linii jest zachowywany przy odczycie. Dodanie \0 na końcu odczytanego ciągu znaków. Zwracany wskaźnik do odczytanego łańcucha lub NULL, gdy błąd lub koniec pliku.

- char *fgets (char *tekst, int dlugosc, FILE *strumien)

Funkcja sformatowanego wejścia analogiczna do scanf( ), odczytująca ze strumienia.

- int fscanf (FILE *strumien, char *format, &arg, ..., &arg)

```
// postać formatu
// % [*] [szerokość][ F | N | h | l | L ]<znak typu>
```

Jeśli podamy jako strumień wejściowy stdin (standardowy strumień wejściowy), to wtedy dane zostaną wczytane z klawiatury.

np. fscanf(stdin, "format" , ...) == scanf("format" , ...)

Funkcja odczytu danych w postaci binarnej (pliki binarne):

```
int  fread (  void *adres_danej,
              size_t  rozmiar_danej,  size_t  ilosc_danych,
              FILE *strumien)

// funkcja odczytująca (ilosc_danych * rozmiar_danej) bajtów
// ze strumienia do wskazanego obszaru pamięci)
// zwraca liczbę odczytanych danych (nie bajtów) lub kod błędu
```

Przykład 3. Odczyt danych z pliku binarnego i tekstowego do tablicy struktur.

```
#include <stdio.h>

const int N=7;
struct  tosoba  {  char  naz[40];   char  data[20];   int  id;  };

void losuj(struct tosoba *s)
{
        sprintf(s->naz,  "naz%d%c", rand() % 99 + 1,0);   //  \0 zero na końcu
        sprintf(s->data, "data%d%c", rand() % 100 + 1900,0);
        s->id = rand() % 99 + 1;
}

void pisz(struct tosoba *s)
{
        printf("%20s ", s->naz);
        printf("%10s ", s->data);
        printf("%8d \n", s->id);
}

void main( void )
{
        FILE *plik;
        struct tosoba mbaza[N], mbaza1[N], mbaza2[N];

printf("Odczyt danych binarnych \n");

 if ( (plik = fopen( "dane.bin" ,  "rb" ) ) != NULL )
 {
    // odczyt zawartości pliku binarnego do tablicy struktur

        fread( mbaza1, sizeof(struct tosoba1), N , plik);
        // lub // fread( mbaza1, sizeof(mbaza1), 1 , plik);

        fclose( plik );

        for (int i=0; i<N; i++) { pisz(&mbaza1[i]); }
  }

  getchar();

printf("Odczyt danych tekstowych \n");

if ( (plik = fopen("dane.txt" ,  "rt") ) != NULL )
 {
         // odczyt zawartości pliku tekstowego do tablicy struktur
        int ile=0;

  for( int i = 0;  i < N+1;  i++ )
  {
     ile=fscanf ( plik,  "%s  %s  %d\n", &mbaza2[ i ].naz, &mbaza2[ i ].data,
                                          &mbaza2[ i ].id );
     if ( (ile<3) || (ile==EOF) ) break;   // czy wczytano 3 dane? lub koniec
        }

   fclose( plik );

   for (int i=0; i<N; i++) { pisz(&mbaza2[i]); }
   }

        }
}
```

Przykład. 4. Testowanie trybu otwarcia tekstowego.

```
void test_trybu_otwarcia_t()
{
   int ile=0;

        FILE *f = fopen("d.txt", "w+t");
        if(f == NULL)
        {
                perror("Error");
                return;
        }

       char buf[4] = {'A', 'B', '\x0a', 'C'};
       ile = fwrite(buf, sizeof(char), 4 ,f);
        // w pliku 5 bajtów:     A B \r \n C
       fclose(f);

      f = fopen("d.txt", "r+t");
        if(f == NULL)
        {  printf("Blad");  return;  }

       printf("\n");
       char z=0;
       do {
        z=getc(f);              // \r pomijany przy odczycie t
        printf("%d\n", z);
       } while (z!=EOF);

       fclose(f);
 }
```

15.7. Wybrane funkcje obsługi plików

Inne funkcje obsługi plików:

Testowanie osiągnięcia końca pliku. Zwracane 0 jeśli koniec pliku lub liczba różna od zera w przeciwnym przypadku.

- int feof (FILE *strumien)

Przesunięcie wskaźnika pliku o zadaną ilość bajtów względem ustalonego miejsca. Zwracane 0 jeśli przesunięcie powiodło się. Kod błędu jeśli nie.

SEEK_SET	- względem początku pliku
SEEK_CUR	- względem aktualnej pozycji
SEEK_END	- względem końca pliku

- int fseek (FILE *strumien, long przes, int wzgledem)

Wyznaczanie w bajtach aktualnego położenia wskaźnika pliku względem jego początku. Zwracane przesunięcie lub kod błędu.

- long ftell (FILE *strumien)

Czyszczenie zawartości bufora wykorzystywanego przez strumień.

- int fflush (FILE *strumien)

15.8. Przykłady obsługi plików

Przykładowe zadania:

Zad. 1. Wykorzystując funkcję fprintf zapisać do pliku dane typu int, char, float oraz daną typu łańcuchowego w wybranym formacie (np. %5d%3c%8.2f%10s; dla danych 23, 'A', 234.57, "Ala_Ola" w pliku fizycznym zapisane zostaną ∇∇∇23∇∇A∇∇234.57∇∇∇Ala_Ola). Odczytać wprowadzone dane za pomocą funkcji fscanf. Należy pamiętać, że podczas odczytu, dla wszystkich formatów oprócz %c, przed zinterpretowaniem są pomijane spacje. W celu pominięcia zbędnych spacji (∇ - symbol spacji) można wykorzystać format %*[∇]%c lub "∇%c".

```
void zad1()
{
	FILE *f = fopen("plik1.txt", "w+");
				// otwarcie do zapisu z możliwością odczytu
				// tryb otwarcia domyślny - tekstowy t
	if (f == NULL)
	{
		printf("Blad");
		return;		// koniec funkcji
	}
	int I = 23;
	char C = 'A';
	float F = 234.57;
	char *S = "Ala_Ola";

	fprintf(f, "%5d%3c%8.2f%10s", I, C, F, S);

	I = 0;
	C = 0;
	F = 0;
	S = "";
	fseek(f, 0L, SEEK_SET);
				// wskaźnik pliku na odległość 0 od początku pliku
	char BUF[20];

	fscanf(f, "%d%*[ ]%c%f%s", &I, &C, &F, BUF);
	printf("%5d%3c%8.2f%10s", I, C, F, BUF);
}
```

Zad. 2. Dany jest typ strukturalny

```
struct tosoba { char naz[20];  unsigned wiek;  long id; }.
```

Opracować następujące funkcje:

a) zapisującą do pliku znakowego łańcuch "-- ... --\n" złożony z 30 znaków '-' i jednego znaku '\n', a następnie pola zmiennej strukturalnej tosoba w wybranym formacie (np. dla fprintf: nazwisko - %20s\n, wiek - %6u\n, identyfikator - %10ld\n);
b) odczytującą z pliku łańcuch złożony z 30 znaków '-' i jednego znaku '\n', a następnie pola struktury typu tosoba rozdzielone znakami przejścia do nowej linii (pominięcie sekwencji "-- ...--\n" o dowolnej długości można zrealizować za pomocą formatu %*[-\n], który oznacza, że znaki '-' oraz '\n' są pomijane; czytanie rozpocznie się jeśli pojawi się znak różny od [-\n]).

W programie głównym zapisać do pliku N struktur (N – stała), a następnie odczytać dane.

```
struct tosoba
{
	char naz[20];   unsigned wiek;    long id;
};
```

```
void zapisz(FILE *f, tosoba s)
{
	fprintf(f, "-------------------------------\n");
	fprintf(f, "nazwisko - %20s\nwiek - %6u\nidentyfikator - %10ld\n",
		s.naz, s.wiek, s.id);		// %s nie zapisuje 0 do pliku
}

void odczytaj(FILE *f, tosoba &s)
{
	fscanf(f, "%*[-\n]nazwisko - %20s\nwiek - %6u\nidentyfikator –
		%10ld\n", s.naz, &s.wiek, &s.id);
	// %s opuszcza spacje i inne białe znaki;  kończy gdy spacja lub
	// inny znak biały;  dodaje 0 na końcu łańcucha
}

void pisz(tosoba s)
{
	printf("naz: %s, ", s.naz);
	printf("wiek: %u, ", s.wiek);
	printf("id: %ld\n", s.id);
}

void losuj(struct tosoba *s)
{
	sprintf(s->naz,  "naz%d%c", rand() % 99 + 1,0);   //  \0 zero na końcu
	s->wiek = rand()%90 + 1;
	s->id = rand() % 99 + 1;
}

void zad2()
{  int i;
	FILE *f = fopen("dane2.txt", "w+");
		// plik tekstowy do zapisu z możliwością odczytu
		// tryb otwarcia tekstowy t
	if (f == NULL)
	{
		printf("Blad");
		return;
	}

	const N = 10;
	tosoba *tab = new tosoba[N];

	for(i = 0; i < N; i++)  losuj(tab[i]);

	printf("Przed zapisaniem:\n");
	for(i = 0; i < N; i++)
	{
		pisz(tab[i]);
		zapisz(f, tab[i]);

		//zamazanie danych
		losuj(tab[i]);
	}

	fseek(f, 0L, SEEK_SET);    // na początek pliku

	for(i = 0; i < N; i++)  odczytaj(f, tab[i]);

	printf("\nPo odczytaniu:\n");
	for(i = 0; i < N; i++)	pisz(tab[i]);

	delete [] tab;
	fclose(f);
}
```

Zad. 3. Dany jest plik tekstowy zawierający liczby rzeczywiste typu double zapisane w formacie z dwoma miejscami po przecinku i rozdzielone spacjami. Napisać program, który:

- zliczy liczbę danych w pliku, czytając dane z pliku aż do napotkania końca pliku lub do pierwszego nieudanego odczytu;
- przewinie plik na początek;
- dokona dynamicznej alokacji tablicy typu double, która pomieści wszystkie dane z pliku;
- wczyta dane z pliku do utworzonej tablicy;
- zamknie plik;
- wyprowadzi dane z tablicy na ekran.

Przed zakończeniem programu zwolnić zaalokowaną pamięć.

Plik z danymi zapisanymi w postaci tekstowej: "liczby.txt".
Kolejne liczby są rozdzielone spacjami.

1 2 3 4 2.34 5.75 6.77 8.99 20 21 13.5

```
void zad3()
{
	FILE *f = fopen("liczby.txt", "r");	// otwarcie tylko do odczytu
	if (f == NULL)  {  printf("Blad");	return;   }

	double d;
	int i = -1;
	int n, c;
	do
	{   i++;   n = fscanf(f, "%lf", &d);
	  }  while (n && (n != EOF));    // EOF = -1

	fseek(f, 0L, SEEK_SET);		// na początek pliku

	double *tab = new double[i];

	for(c = 0; c < i; c++)  fscanf(f, "%lf", &tab[c]);

	fclose(f);

	for(c = 0; c < i; c++) printf("%.2lf\n", tab[c]);

	delete [] tab;
}
```

Zad. 4. Opracować program, który na podstawie rozmiaru pliku binarnego określi ile zawiera on liczb typu long, a następnie utworzy dynamiczną tablicę liczb typu long i wczyta do niej zawartość pliku. Wyprowadzić zawartość tablicy na ekran. Przed zakończeniem programu zwolnić pamięć. Uwaga: rozmiar pliku można określić za pomocą funkcji ftell.

Plik z danymi zapisanymi w postaci binarnej (liczby typu long): ”dane.dat”. Dane zapisane do pliku w trybie binarnym.

```
void zad4()
{
	FILE *f = fopen("dane.dat", "rb");
				// otwarcie tylko do odczytu
			       // tryb otwarcia binarny b
	if (f == NULL)
	{
		printf(”Blad”);   return;
	}
	fseek(f, 0L, SEEK_END);	// pozycja 0 od końca pliku
	long len = ftell(f);		// rozmiar pliku
	fseek(f, 0L, SEEK_SET);	// na początek pliku

	int n = len / sizeof(long);
	long c;

	long *tab = new long[n];
	long temp;
	for(c = 0; c < n; c++)
	{
		fread(&temp, sizeof(long), 1, f);	// odczyt danych
		printf("%ld\n", temp);
	}

	delete [] tab;
	fclose(f);
}
```