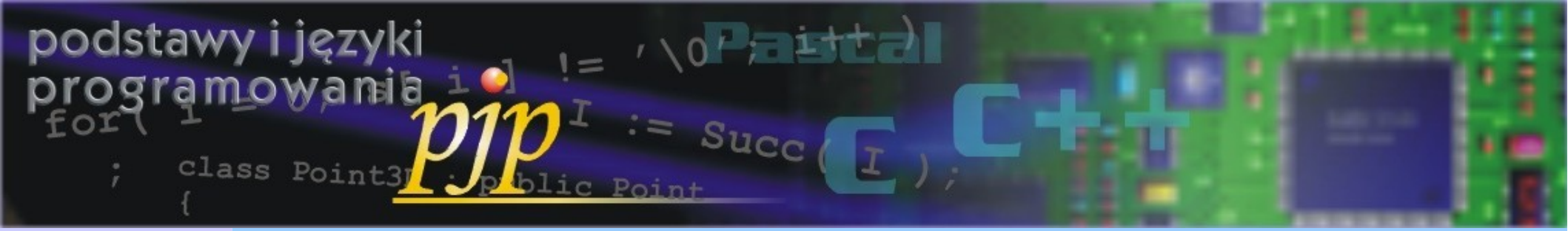


Podstawy programowania w języku C++

Część dwunasta

Przetwarzanie plików amorficznych
Konwencja języka C

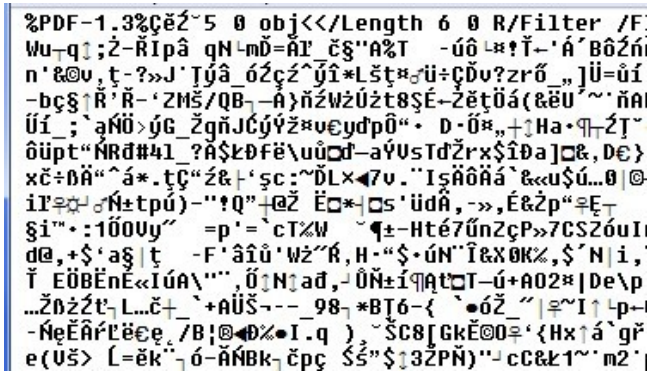
Autor
Roman Simiński

Kontakt
roman.siminski@us.edu.pl
www.programowanie.siminskionline.pl

Przetwarzanie plików binarnych, motywacja

- Często pliki nie zawierają danych tekstowych. Przykładem są pliki graficzne, dźwiękowe czy multimedialne.
- Ich zawartość to najczęściej *binarny* obraz zawartości pamięci operacyjnej (np. ciąg bajtów opisujących kolory piksela) uzupełniony o dodatkowe informacje (np. nagłówek pliku BMP, czy dane EXIF).
- Do przetwarzania plików, których zawartość ma charakter binarny, wykorzystuje się najczęściej *odczyt/zapis bloków*.
- Pod pojęciem *bloku* rozumieć będziemy *ciąg bajtów o określonej długości*, nie zakładamy, że ciąg taki ma jakakolwiek strukturę. *Rozmiar* takiego bloku jest określony *liczbą jego bajtów*.

Plik PDF w podglądzie tekstowym

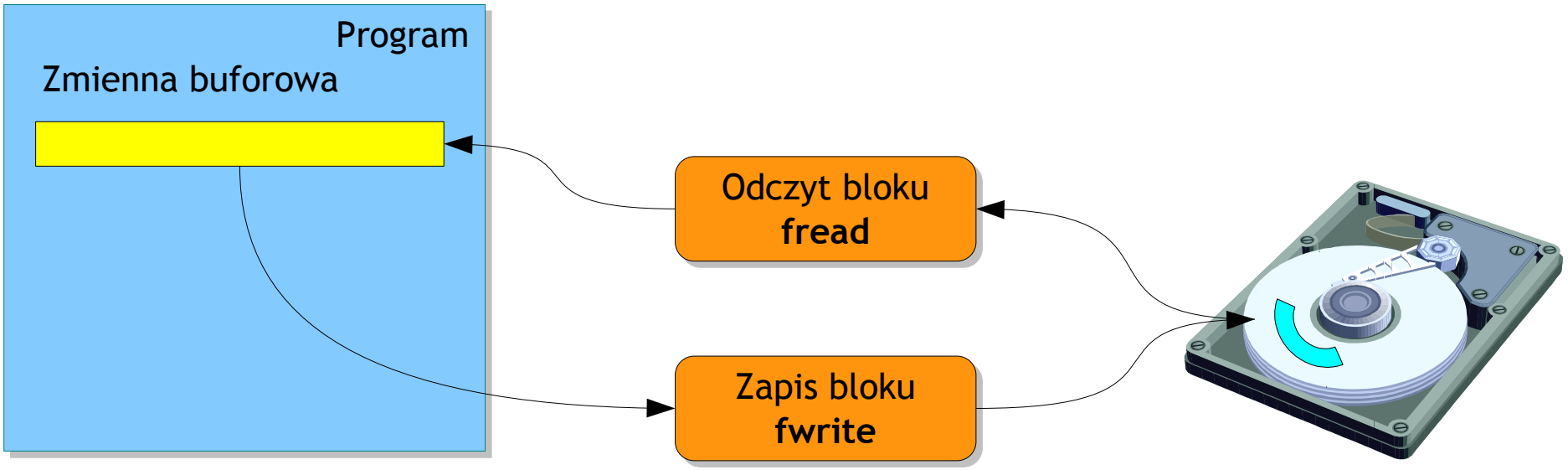


Plik JPG w podglądzie tekstowym

Przetwarzanie plików binarnych, koncepcja

Aby korzystać z blokowego odczytu i zapisu musimy spełnić dwa warunki:

- musimy dysponować otwartym plikiem, zaleca się, aby plik taki otwarty był w trybie binarnym;
- musimy w programie posiadać zmienną — która realizuje funkcję bufora — z której będą pobierane dane do zapisu, lub do której będą pobierane dane w przypadku odczytu.



Przetwarzanie plików binarnych, przykład

Załóżmy, że chcemy napisać program, którego zadaniem jest:

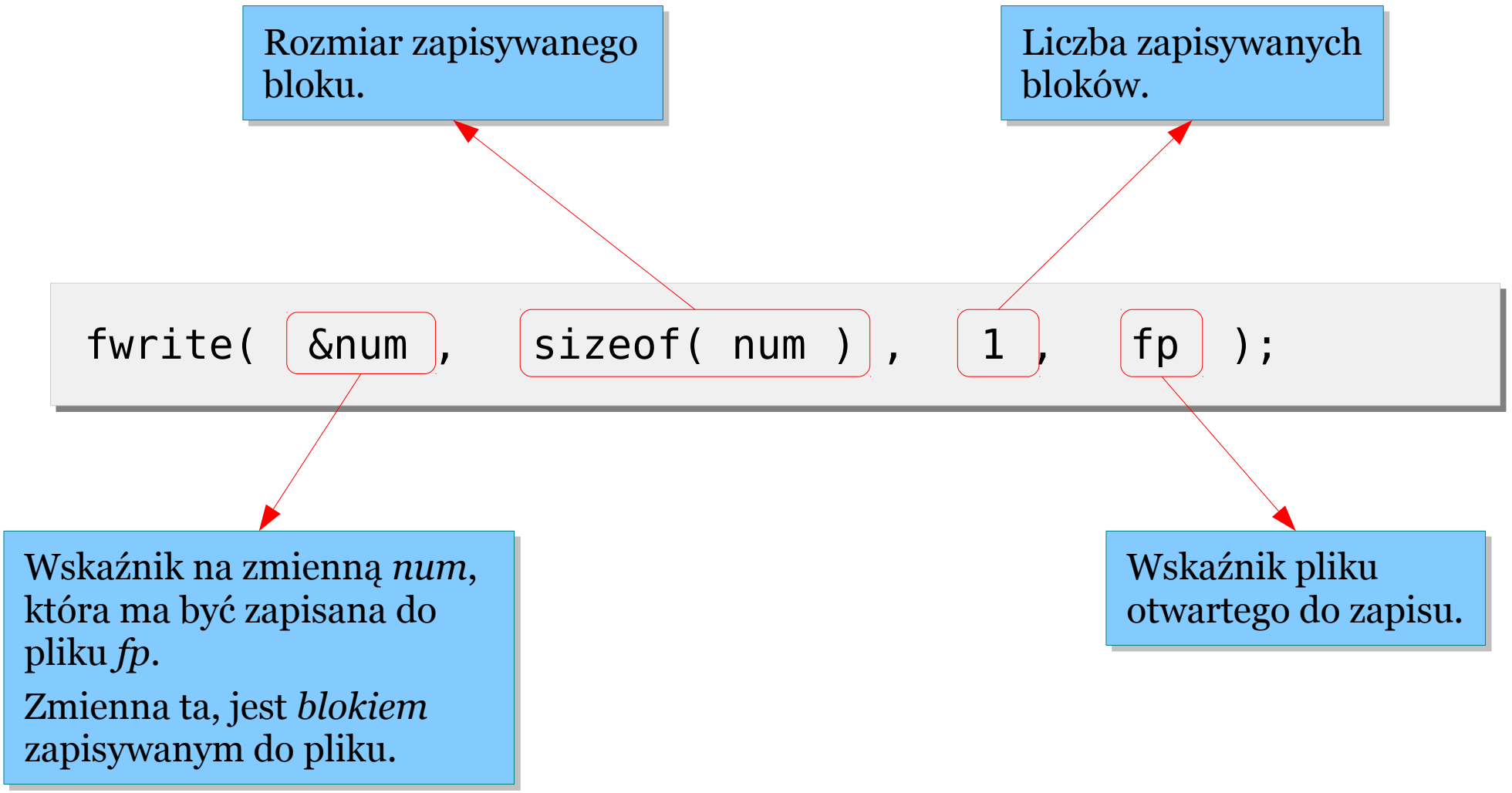
- utworzenie nowego pliku binarnego, zapisanie do niego liczby typu *float* o wartości 123.321, zamknięcie pliku;
- powtórne jego otwarcie w trybie do odczytu, odczytanie zapisanej wcześniej liczby i wyprowadzenie jej do *stdout*.

```
Zapis liczby: 123.321
Odczyt liczby: 123.321

Nacisnij Enter by zakonczyc...
```

Przetwarzanie plików binarnych, otwarcie pliku, zapis liczby typu float

```
#include <cstdio>
#include <cstdlib>

int main()
{
  FILE * fp;
  float num = 123.321;

  if( ( fp = fopen( "d.dat", "wb" ) ) != NULL )
  {
     cout << "\nZapis liczby: " << num;
     fwrite( &num, sizeof( num ), 1, fp );
     fclose( fp );
  }

  .
  .
  .

  return EXIT_SUCCESS;
}
```

Przetwarzanie plików binarnych, zapis zmiennej num



Przetwarzanie plików binarnych, otwarcie pliku, zapis liczby typu float

```
if( ( fp = fopen( "d.dat", "wb" ) ) != NULL )
{
    printf( "\nZapis liczby: %g", num );
    fwrite( &num, sizeof( num ), 1, fp );
    fclose( fp );
}
```

Watch 4

```
sizeof( num ): 4
num, M: 5A A4 F6 42
```

Output 3

```
Zapis liczby: 123.321
```

Name	Ext	Size
[..]		<DIR>
D	DAT	4

Zawartość pliku d.dat

```
00000000  5A A4 F6 42
```

```
1  Z¤öB
```

Szesnastkowo

Jako tekst

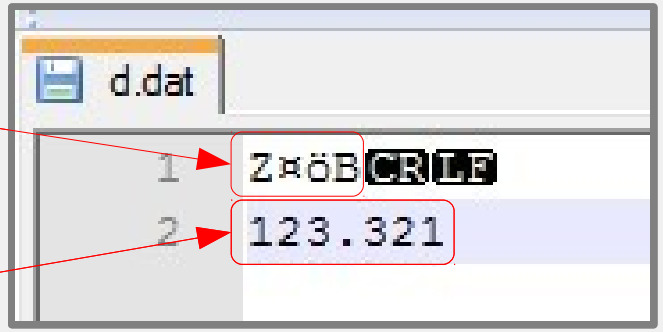
Opis funkcji blokowego zapisu — fwrite

```
size_t fwrite( void * ptr, size_t size, size_t n, FILE * stream );
```

- Funkcja zapisuje dane z obszaru pamięci wskazywanego przez *ptr* do strumienia *stream*.
- Zapisuje *n* bloków o rozmiarze *size*.
- Łączna liczba zapisanych bajtów to *n*size*.
- Rezultatem funkcji jest liczba *zapisanych bloków* (nie bajtów!).
- W przypadku wystąpienia *końca pliku* lub *błędu*, rezultatem funkcji jest liczba, potencjalnie zerowa, *bezbłędnie zapisanych bloków*.

Zapis liczby jako tekstu a zapis jej binarnej reprezentacji

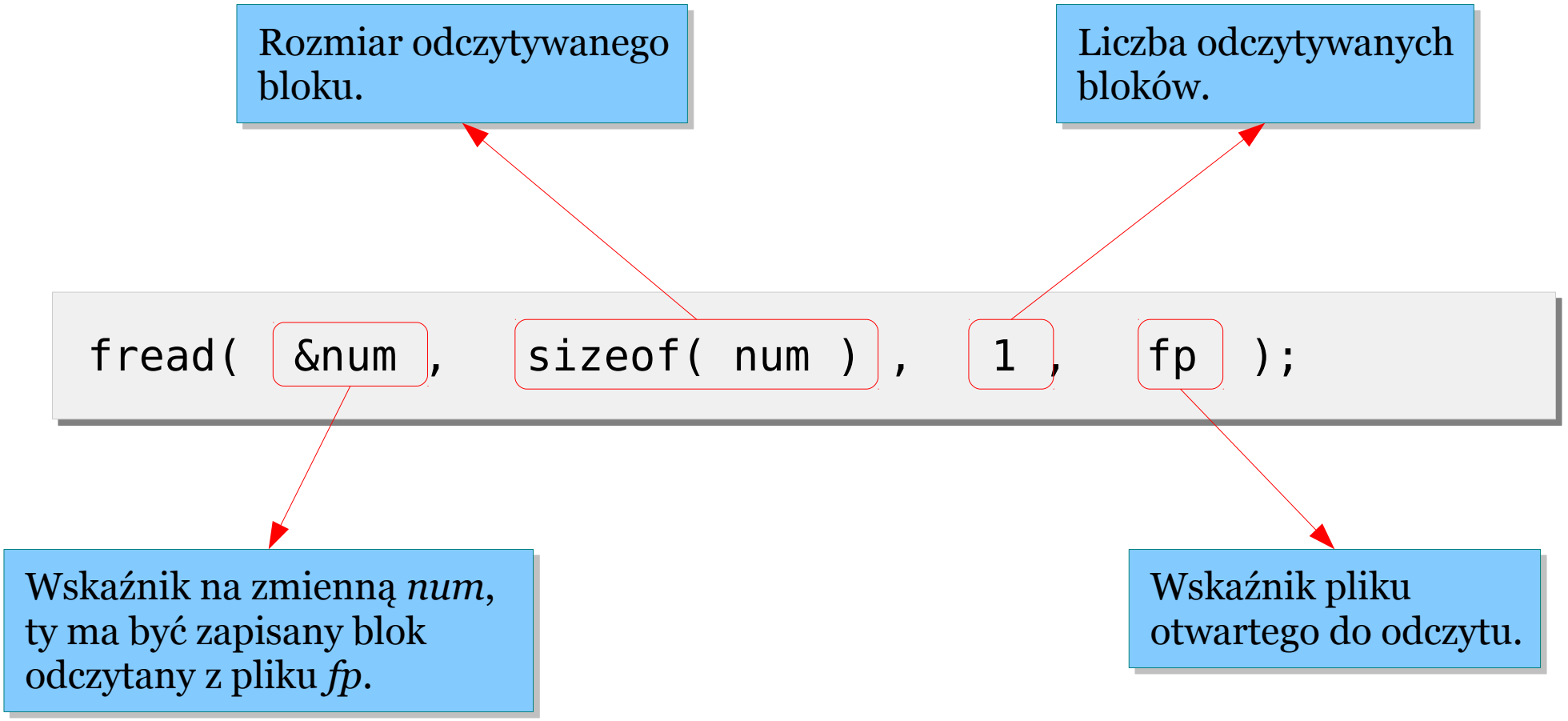
```
int main()
{
  FILE * fp;
  float num = 123.321;

  if( ( fp = fopen( "d.dat", "wt" ) ) != NULL )
  {
    fwrite( &num, sizeof( num ), 1, fp );

    fputc( '\n', fp );

    fprintf( fp, "%g", num );

    fclose( fp );
  }
  . . .
}
```



Przetwarzanie plików binarnych, otwarcie pliku, odczyt liczby typu float

```
#include <cstdio>
#include <cstdlib>

int main()
{
  FILE * fp;
  float num = 123.321;
  .
  .
  .
  num = 0;
  if( ( fp = fopen( "d.dat", "rb" ) ) != NULL )
  {
     fread( &num, sizeof( num ), 1, fp );
     cout << "\nOdczyt liczby: " << num;
     fclose( fp );
  }

  cout << "\n\nNacisnij Enter by zakonczyc...";
  ( void )getchar();
  return EXIT_SUCCESS;
}
```

Przetwarzanie plików binarnych, odczyt do zmiennej num



Opis funkcji blokowego odczytu — fread

```
size_t fread( void * ptr, size_t size, size_t n, FILE * stream );
```

- Funkcja czyta dane ze strumienia *stream* do obszaru pamięci wskazywanego przez *ptr*.
- Odczytuje *n* bloków o rozmiarze *size*.
- Łączna liczba odczytanych bajtów to *n*size*.
- Rezultatem funkcji jest liczba *przeczytanych bloków* (nie bajtów!).
- W przypadku napotkania *końca pliku* lub *błędu*, rezultatem jest liczba bezbłędnie *odczytanych bloków*, która potencjalnie może być równa zero.

Odczyt i zapis z kontrolą poprawności

- Funkcje *fread* i *fwrite* pozwalają na kontrolę poprawności wykonywanych operacji odczytu i zapisu.
- Wystarczy kontrolować rezultat wywołania tych funkcji i porównywać z liczbą określonych bloków.

```
if( ( fp = fopen( "d.dat", "wb" ) ) != NULL )
{
   if( fwrite( &num, sizeof( num ), 1, fp ) != 1 )
     cout << "\nBlad zapisu!";
   else
     cout << "\nZapis wykonany";
   fclose( fp );
}
```

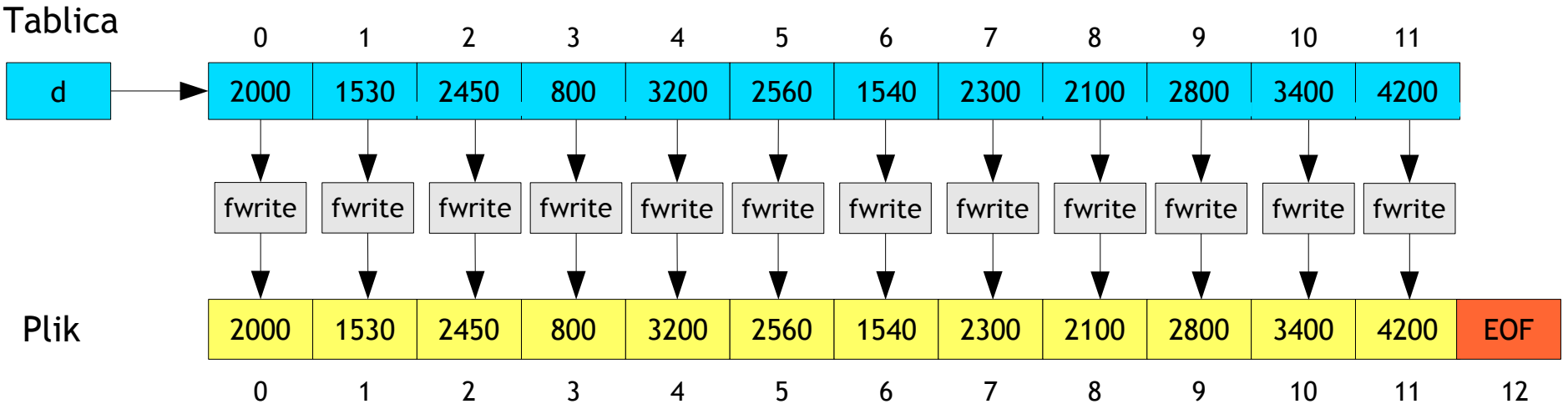
```
if( ( fp = fopen( "d.dat", "rb" ) ) != NULL )
{
   if( fread( &num, sizeof( num ), 1, fp ) != 1 )
     cout << "\nBlad odczytu!";
   else
     cout << "\nOdczyt liczby: " << num;
   fclose( fp );
}
```

Zapis i odczyt ciągów danych

- Załóżmy, że zapisujemy do pliku 12-cie liczb typu *float* reprezentujących dochody z kolejnych miesięcy roku podatkowego.
- Dane źródłowe są zapisane w dwunastoelementowej tablicy o nazwie *d*:

```
const int LB_MIES = 12;
. . .
float d[ LB_MIES ];
```

- Pierwszym narzucającym się rozwiązaniem jest zapisanie kolejno każdego elementu tablicy jako bloku, wykorzystując funkcję *fwrite*.



Zapis i odczyt ciągów danych, przykład 1-szy

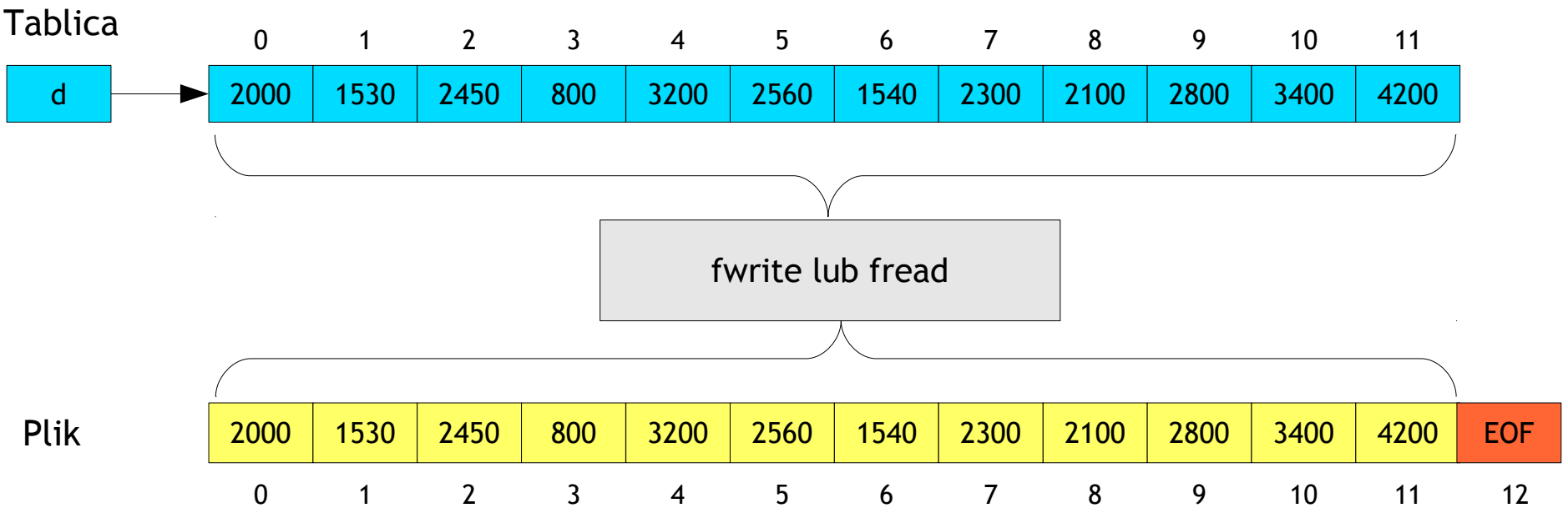
```
#include <cstdio>
#include <cstdlib>
const int LB_MIES = 12

int main()
{
  FILE * fp;
  float d[ LB_MIES ];
  int nr;

  // Wstawiamy do tablicy przykładowe dane
  for( nr = 0; nr < LB_MIES; nr++ )
    d[ nr ] = 1000 * ( nr + 1 );

  // Zapis tablicy d, element po elemencie, do pliku d.dat
  if( ( fp = fopen( "d.dat", "wb" ) ) != NULL )
  {
     for( nr = 0; nr < LB_MIES; nr++ )
       if( fwrite( &d[ nr ], sizeof( d[ nr ] ), 1, fp ) != 1 )
         cout << "\nBlad zapisu!";
       else
         cout << "\nZapisano: " << d[ nr ];
     fclose( fp );
  }
  . . .
```

Zapis i odczyt ciągów danych, przykład 1-szy, cd ...

```
  . . .
  // Zerujemy tablice by stwierdzic czy odczyt dziala
  for( nr = 0; nr < LB_MIES; nr++ )
    d[ nr ] = 0;

  // Odczyt danych z d.dat do tablicy d, element po elemencie
  if( ( fp = fopen( "d.dat", "rb" ) ) != NULL )
  {
     for( nr = 0; nr < LB_MIES; nr++ )
       if( fread( &d[ nr ], sizeof( d[ nr ] ), 1, fp ) != 1 )
         cout << "\nBlad odczytu!";
       else
         cout << "\nOdczytano: " << d[ nr ];
     fclose( fp );
  }

  cout << "\n\nNacisnij Enter by zakonczyc...";
  ( void )getchar();
  return EXIT_SUCCESS;
}
```

Zapis i odczyt ciągów danych, rozwiązanie 2-gie

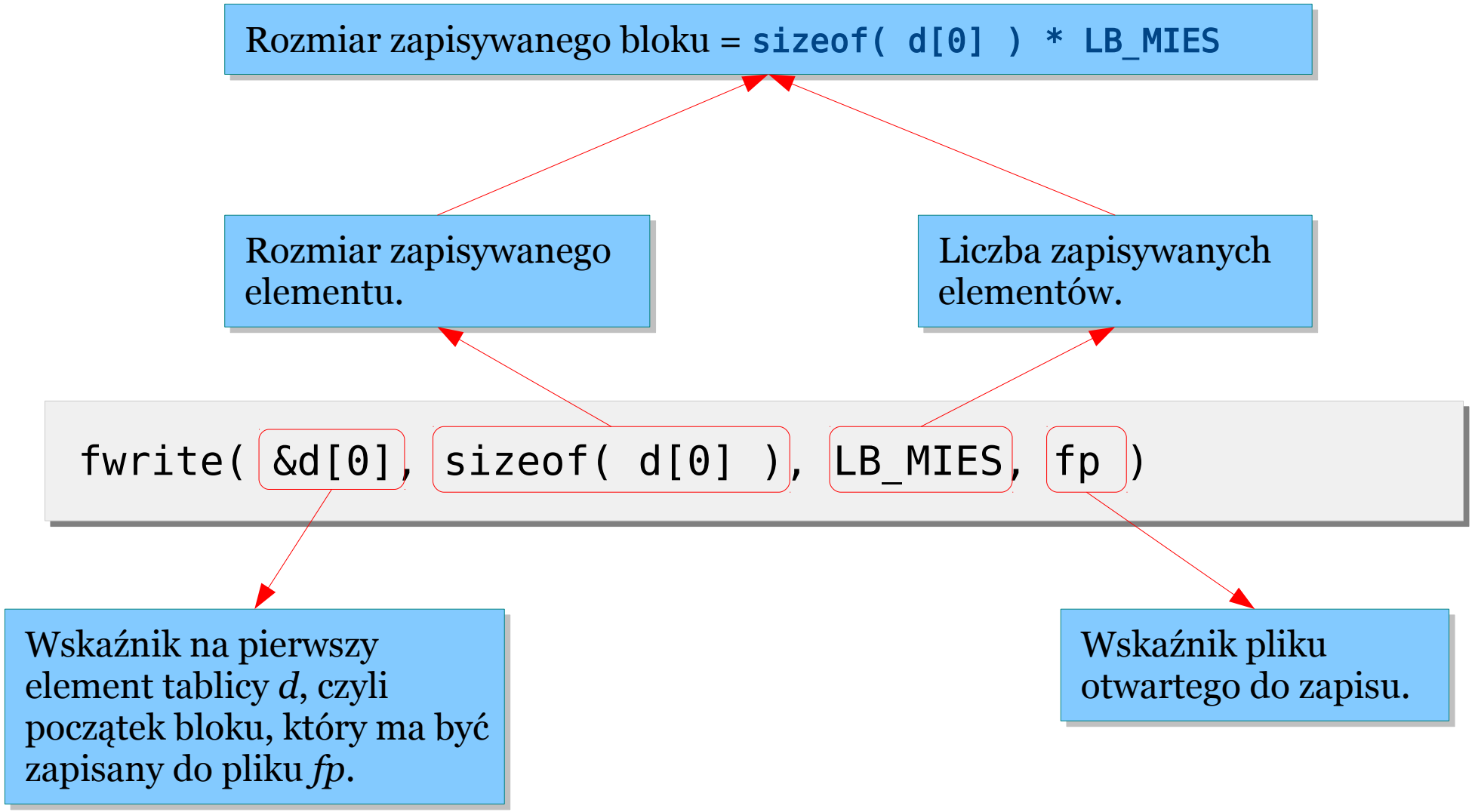
- Cała tablica może być blokiem, zapisywanym/odczytywanym jednym wywołaniem instrukcji *fwrite/fread*.



Zapis i odczyt ciągów danych, przykład 2-gi

```
#include <cstdio>
#include <cstdlib>
const int LB_MIES = 12

int main()
{
  FILE * fp;
  float d[ LB_MIES ];
  int nr;

  // Wstawiamy do tablicy przykładowe dane
  for( nr = 0; nr < LB_MIES; nr++ )
    cout << "\nZapis: " << ( d[ nr ] = 1000 * ( nr + 1 ) );


  if( ( fp = fopen( "d.dat", "wb" ) ) != NULL )
  {
     if( fwrite( &d[0], sizeof( d[0] ), LB_MIES, fp ) != LB_MIES )
       cout << "\nBlad zapisu!";
     fclose( fp );
  }

  . . .
```

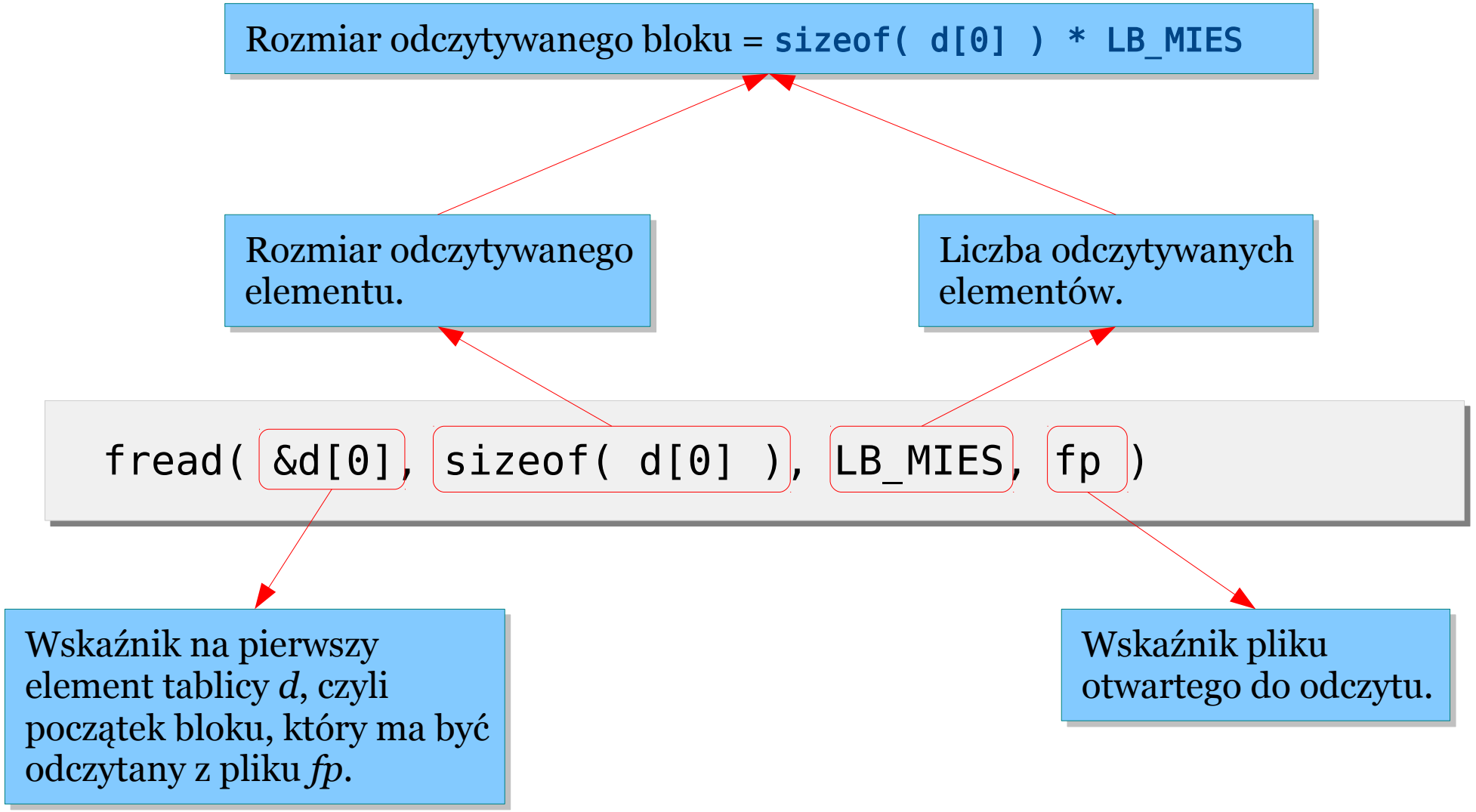
Zapis i odczyt ciągów danych, przykład 2-gi



Zapis i odczyt ciągów danych, przykład 2-gi

```
  . . .

  // Zerujemy tablice by stwierdzic czy odczyt dziala
  for( nr = 0; nr < LB_MIES; nr++ )
    d[ nr ] = 0;

  if( ( fp = fopen( "d.dat", "rb" ) ) != NULL )
  {
     if( fread( &d[0], sizeof( d[0] ), LB_MIES, fp ) != LB_MIES )
       cout << "\nBlad odczytu!";
     fclose( fp );
  }

  for( nr = 0; nr < LB_MIES; nr++ )
    cout << "\nOdczyt:” << d[ nr ];

  cout << "\n\nNacisnij Enter by zakonczyc...";
  ( void )getchar();
  return EXIT_SUCCESS;
}
```

Zapis i odczyt ciągów danych, przykład 2-gi



Zapis i odczyt ciągów danych, uzupełnienie

- Nazwa tablicy jest ustalonym wskaźnikiem na jej poczatek, czyli na pierwszy element.
- Zatem zamiast **`&d[ 0 ]`** można napisać po prostu **`d`**:

```
fread( &d[0], sizeof( d[0] ), LB_MIES, fp )
```



```
fread( d, sizeof( d[0] ), LB_MIES, fp )
```

Zapis i odczyt blokowy — dana typu int

```
int zmiennaInt = 10;
FILE * fp;
. . .
if( fwrite( &zmiennaInt, sizeof( zmiennaInt ), 1, fp ) != 1 )
  cout << "\nBlad zapisu!";
else
  cout << "\nZapisano liczbe: " << zmiennaInt;
```

Możemy napisać funkcję, realizującą zapis pojedynczej danej typu *int*:

```
int zmiennaInt = 10;
FILE * fp;
. . .
bool writeInt( int i, FILE * f )
{
  return fwrite( &i, sizeof( i ), 1, f ) == 1;
}

if( !writeInt( zmiennaInt, fp ) )
  cout << "\nBlad zapisu!";
else
  cout << "\nZapisano liczbe: " << zmiennaInt;
```

Zapis i odczyt blokowy — dana typu float

```
float zmiennaFloat = 10;
FILE * fp;
. . .
if( fwrite( &zmiennaFloat, sizeof( zmiennaFloat ), 1, fp ) != 1 )
  cout << "\nBlad zapisu!";
else
  cout << "\nZapisano liczbe: " << zmiennaFloat;
```

Możemy napisać funkcję, realizującą zapis pojedynczej danej typu *float*:

```
float zmienna_float = 10;
FILE * fp;
. . .
bool write_float( float n, FILE * f )
{
  return fwrite( &n, sizeof( n ), 1, f ) == 1;
}

if( ! write_float( zmienna_float, fp ) )
  cout << "\nBlad zapisu!";
else
  cout << "\nZapisano liczbe: " << zmiennaFloat;
```

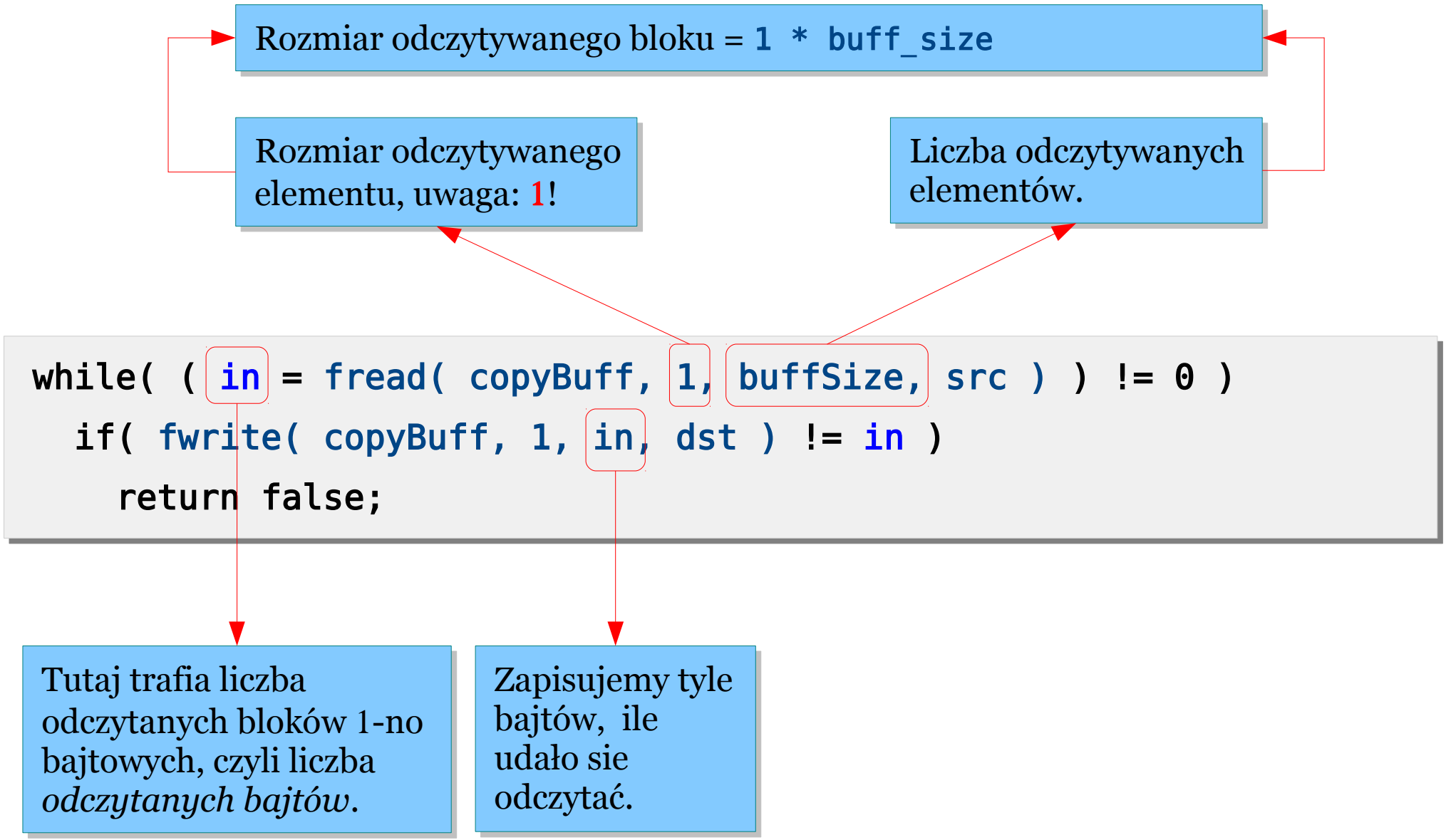
Warto napisać sobie zestaw przydatnych funkcji

```
. . .
bool writeInt( int n, FILE * f )
{
  return fwrite( &n, sizeof( n ), 1, f ) == 1;
}

bool writeFloat( float n, FILE * f )
{
  return fwrite( &n, sizeof( n ), 1, f ) == 1;
}

bool writeDouble( double n, FILE * f )
{
  return fwrite( &n, sizeof( n ), 1, f ) == 1;
}

bool writeWord( unsigned short int n, FILE * f )
{
  return fwrite( &n, sizeof( n ), 1, f ) == 1;
}

. . .
```

Kopiowanie zawartości plików blok po bloku

```
/*-----------------------------------------------------------------
  Funkcja bpbFileCopy realizuje kopiowanie zawartości źrodłowego
  pliku src do pliku docelowego dst. Wykorzystywane są blokowe
  operacje zapisu i odczytu. Funkcja nie zamyka strumieni src i dst.
  Parametry : Wskaźniki na prawidłowo otwarte strumienie binarne
              src, dst - odpowiednio dla pliku źródlowego i docelowego.
  Rezultat  : true  jeżeli kopiowanie zakończyło się poprawnie
              false jeżeli wystąpił błąd podczas kopiowania
-----------------------------------------------------------------*/
int bpbFileCopy( FILE * dst, FILE * src )
{
  char * copyBuff = 0;                // Wskaźnik na bufor kopiowania
  size_t buffSize = 30 * 1024;        // Rozmiar bufora kopiowania
  size_t in = 0;                      // Liczba przeczytanych bloków

  if( ( copyBuff = new (nothrow) char[ buffSize ] ) == 0 )
    return false;

  while( ( in = fread( copyBuff, 1, buffSize, src ) ) != 0 )
    if( fwrite( copyBuff, 1, in, dst ) != in )
      return false;

  delete [] copyBuff;
  return true;
}
```

Uwaga, algorytm wykorzystuje drobny trik



Wyświetlanie zawartości pliku w widoku: szesnastkowo-ASCII

Jakiś plik o dowolnej zawartości:

```
  while( ( in_chars = fread( buffer, 1, BUFFER_LEN, file ) ) > 0  )
  {
      /* Wypisz : hex, dwie pozycje, wiodące zera, duże litery */
      for( i = 0; i < in_chars; i++)
         printf( "%02X ", buffer[ i ] );

      printf("| ");   /* Separator części szesnastkowej od ASCII */

      /* Wypisz bufor jako ASCII o ile można, jeśli nie to '.' */
```

```
29 0D 0A 20 20 20 20 20 20 20 20 20 72 69 6E 74 66 28 | )..          printf(
20 22 25 30 32 58 20 22 2C 20 62 75 66 66 65 72 5B 20 69 | "%02X ", buffer[ i
20 5D 20 29 3B 0D 0A 0D 0A 20 20 20 20 20 20 70 72 69 6E | ] );....      prin
74 66 28 22 7C 20 22 29 3B 20 20 2F 2A 20 53 65 70 61 72 | tf("| ");  /* Separ
61 74 6F 72 20 63 7A A9 98 63 69 20 73 7A 65 73 6E 61 73 | ator cz..ci szesnas
74 6B 6F 77 65 6A 20 6F 64 20 41 53 43 49 49 20 2A 2F 0D | tkowej od ASCII */.
0A 0D 0A 20 20 20 20 20 20 20 2F 2A 20 57 79 73 77 69 65 74 | ...      /* Wyswiet
6C 20 62 75 66 6F 72 20 6A 61 6B 6F 20 41 53 43 49 49 20 | l bufor jako ASCII
6F 20 69 6C 65 20 6D 6F 7A 6E 61 2C 20 6A 61 6B 20 6E 69 | o ile mozna, jak ni
65 20 74 6F 20 77 79 73 77 69 65 74 6C 20 27 2E 27 20 2A | e to wyswietl '.' *
2F 0D 0A 20 20 20 20 20 20 20 66 6F 72 28 20 69 20 3D 20 30 | /..      for( i = 0
3B 20 69 20 3C 20 69 6E 5F 63 68 61 72 73 3B 20 69 2B 2B | ; i < in_chars; i++
20 29 0D 0A 20 20 20 20 20 20 20 20 20 20 70 72 69 6E 74 66 | ).          printf
28 22 25 63 22 2C 20 69 73 70 72 69 6E 74 28 20 62 75 66 | ("%c", isprint( buf
66 65 72 5B 20 69 20 5D 20 29 20 3F 20 62 75 66 66 65 72 | fer[ i ] ) ? buffer
5B 20 69 20 5D 20 3A 20 27 2E 27 20 29 3B 0D 0A 0D 0A 20 | [ i ] : '.' );....
20 20 20 20 20 70 75 74 63 68 61 72 28 27 5C 6E 27 29 3B |      putchar('\n');
0D 0A 0D 0A 20 20 20 20 20 20 20 69 66 28 20 28 20 2B 2B 6C | ....       if( ( ++l
69 6E 65 73 20 25 20 50 41 47 45 5F 4C 45 4E 47 54 48 20 | ines % PAGE_LENGTH
29 20 3D 3D 20 30 20 29 20 20 2F 2A 20 43 7A 79 20 65 6B | ) == 0 )  /* Czy ek
72 61 6E 20 7A 61 70 65 88 6E 69 6F 6E 79 3F 20 2A 2F 0D | ran zape.niony? */.
```

Wyświetlanie zawartości pliku w widoku: szesnastkowo-ASCII

```
/*-------------------------------------------------------------------
  Funkcja hex_dump wyprowadza do stdout zawartość pliku wyświetlaną
  w postaci szesnastkowej oraz ASCII.
  Parmetry : file - Wskaźnik na prawidłowo otwarty strumień binarny
  Uwaga – funkcja nie zatrzymuje wyświetlania np. co 24 linie.
-------------------------------------------------------------------*/
void hex_dump( FILE * file )
{
  #define BUFFER_LEN 19          /* Tyle znaków będzie w linii na ekranie */
  unsigned char buffer[ BUFFER_LEN ];      /* Bufor na odczytywane znaki */
  int i = 0;

  while( ( in_chars = fread( buffer, 1, BUFFER_LEN, file ) ) > 0  )
  {
      /* Wypisz : hex, dwie pozycje, wiodące zera, duże litery */
      for( i = 0; i < in_chars; i++)
         printf( "%02X ", buffer[ i ] );

      printf("| ");   /* Separator części szesnastkowej od ASCII */

      /* Wypisz bufor jako ASCII o ile można, jeśli nie to '.' */
      for( i = 0; i < in_chars; i++ )
        printf( "%c", isprint( buffer[ i ] ) ? buffer[ i ] : '.' );
      putchar('\n');
   }
}
```