Politechnika Wrocławska

Formatowane (tekstowe) wejście/wyjście. Binarne wejście/wyjście.

wer. 10 z drobnymi modyfikacjami!

Wojciech Myszka

Katedra Mechaniki, Inżynierii Materiałowej i Biomedycznej

2019-05-21 21:01:05 +0200

HR EXCELLENCE IN RESEARCH

Część I

Formatowane (tekstowe) wejście/wyjście

Otwarcie pliku I

1. Przed skorzystaniem z pliku należy go „otworzyć” (*open*).
2. **stdin**, **stdout**, **stderr** otwarte są w trybie tekstowym.
3. Standardowa biblioteka (stdio.h) zawiera trzy funkcje fopen, freopen i fclose .
4. Pierwsze dwie służą do „skojarzenia” (powiązania) pliku z odpowiednią strukturą danych.
5. Ostatnia — powiązanie to likwiduje.
6. Wszystkie czynności związane z otwieraniem i zamykaniem plików realizowane są przez System Operacyjny.

Operację otwarcia pliku (ponownego otwarcia) realizują funkcje **fopen** i **freopen**:

Otwarcie pliku II

```c
#include <stdio.h>
FILE *fopen(const char *filename, const char *mode);
FILE *freopen(const char *filename, const char *mode,
              FILE *stream);
```

Otwarcie pliku III

Użycie:

```
FILE *fp;
...
fp = fopen("plik.txt", "w");
...
fp = freopen("plik.txt", "r", fp);
```

- Pierwszy parametr (filename) to wskaźnik do tablicy tekstowej (na przykład stałej) zawierającej nawę pliku.
- Parametr drugi to ciąg znaków (lub zmienna) zawierająca w sposób symboliczny określenie trybu dostępu:
  - **r** otwórz plik **tekstowy** do odczytu (plik musi istnieć)
  - **w** otwórz plik **tekstowy** do zapisu (niszcząc jego zawartość jeżeli już istnieje)

Politechnika Wrocławska

Otwarcie pliku IV

- **a** (*append*); otwórz plik **tekstowy** do dopisywania, jeżeli plik nie istnieje — zostanie utworzony.
- **r+** otwarcie pliku **tekstowego** do zapisu i odczytu (plik powinien już istnieć!)
- **w+** otwórz istniejący plik (kasując jego zawartość) lub utwórz nowy plik **tekstowy** w trybie do zapisu i odczytu
- **a+** otwórz plik **tekstowy** na końcu w trybie do zapisu i odczytu

Otwarcie pliku V

Przykład

```
#include <stdio.h>
...
FILE *pp;
...
pp = fopen("Ala.txt", "r");
if (pp == NULL)
    { /* Błąd! */}
...
fscanf(pp, "%d", &i);
```

- Trzeci parametr (stream — tylko w przypadku funkcji reopen) zawiera wskaźnik do struktury danych opisujących strumień danych.

Politechnika Wrocławska

Otwarcie pliku VI

1. Funkcja **fopen()** otwiera plik o podanej nazwie we wskazanym trybie, tworzy strukturę danych opisującą go i zwraca do niej adres. W przypadku błędu — zwraca NULL.
2. Funkcja **freopen()** najpierw zamyka otwarty wcześniej plik i otwiera go ponownie we wskazanym trybie, zwracając wskaźnik do struktury danych opisujących strumień. W przypadku błędu — zwraca NULL.
3. Funkcja **fclose()** zamyka wskazany plik. W przypadku błędu — zwraca NULL.

```
#include <stdio.h>
...
int fclose(FILE *stream);
```

stream to wskaźnik do struktury danych opisujących strumień danych. Przykład:

Otwarcie pliku VII

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

Odczyt formatowany I

1. Funkcje typu **scanf**

```
#include <stdio.h>
int fscanf(FILE *stream, const char *format, ...)
    ;

int scanf(const char *format, ...);

int sscanf(const char *s, const char *format,
    ...);
```

Dane czytane są ze wskazanego strumienia (stream) i interpretowane zgodnie z użytym formatem. W formacie powinna wystąpić wystarczająca liczba specyfikacji aby dostarczyć dane dla wszystkich argumentów.

Odczyt formatowany II

- fscanf zwraca liczbę przeczytanych wartości lub wartość EOF
- scanf jest odpowiednikiem fscanf, ale dotyczy strumienia stdin
- sscanf jest odpowiednikiem fscanf, z tym, że dane „czytane" są z tablicy znakowej; dotarcie do końca tabeli jest równoważne z dotarciem do końca pliku

Wejście: Specyfikacja formatu I

Na specyfikację formatu składają się:

- odstępy; Wystąpienie w formacie „białego znaku" (*white space*) powoduje, że funkcje z rodziny scanf będą odczytywać i odrzucać znaki, aż do napotkania pierwszego znaku nie będącego białym znakiem.
- znaki (różne od % i odstępów). Jeżeli taki znak wystąpi w specyfikacji musi się pojawić w strumieniu wejściowym na odpowiednim miejscu!
- specyfikacje konwersji (rozpoczynające się znakiem %)

Po znaku % wystąpić może:

- nieobowiązkowy znak $*$ (oznaczający, że zinterpretowana wartość ma być zignorowana)
- nieobowiązkowa specyfikacja długości pola (określa maksymalną liczbę znaków pola)

Wejście: Specyfikacja formatu II

- jeden z nieobowiązkowych znaków **h**, **l** (mała litera „el") lub **L** mówiących jak ma być interpretowana czytana wielkość (h — **short**, l — **long** albo **double** w przypadku **float**, ll – **long long**, L — **long double**),
- znak określający typ konwersji.

Po znaku procent (%) wystąpić może jeden ze znaków

- d liczba całkowita,
- i liczba całkowita (można wprowadzać wartości szesnastkowe — jeżeli poprzedzone znakami 0x lub ósemkowe jeżeli poprzedzone 0; 031 czytane z formatem %d to 31 a z formatem %i to 25),
- o liczba całkowita kodowana ósemkowo,
- u liczba całkowita bez znaku (**unsigned int**),
- x liczba całkowita kodowana szesnastkowo,

Wejście: Specyfikacja formatu III

- a, e, f, g liczba typu float; można również wczytać w ten sposób nieskończoność lub wartość Not a Number (NaN),
- s ciąg znaków (na końcu zostanie dodany znak zero),
- c znak (lub ciąg znaków gdy wyspecyfikowano szerokość pola), nie jest dodawane zero na końcu,
- [odczytuje niepusty ciąg znaków, z których każdy musi należeć do określonego zbioru, argument powinien być wskaźnikiem na **char**,
- n do zmiennej odpowiadającej tej specyfikacji konwersji wpisana zostanie liczba znaków przetworzonych przez fscanf,
- p w zależności od implementacji — służy do wprowadzania wartość wskaźników,
- % znak %.

Politechnika Wrocławska

Wejście: Specyfikacja formatu IV

Format „[" — Po otwierającym nawiasie następuje ciąg określający znaki jakie mogą występować w odczytanym napisie i kończy się on nawiasem zamykającym tj.]. Znaki pomiędzy nawiasami (tzw. *scanlist*) określają możliwe znaki, chyba że pierwszym znakiem jest ^ — wówczas w odczytanym ciągu znaków mogą występować znaki nie występujące w scanlist. Jeżeli sekwencja zaczyna się od [] lub [^] to ten pierwszy nawias zamykający nie jest traktowany jako koniec sekwencji tylko jak zwykły znak. Jeżeli wewnątrz sekwencji występuje znak − (minus), który nie jest pierwszym lub drugim jeżeli pierwszym jest ^ ani ostatnim znakiem zachowanie jest zależne od implementacji.

Politechnika Wrocławska

Wejście: Specyfikacja formatu V

Przykład

```
#include <stdio.h>
int main(void)
{
    int x, y, n;
    x = scanf("%*d␣%d␣%n", &y, &n);
    printf("y=␣%d,␣n=␣%d\n", y, n);
    printf("x=␣%d\n", x);
    return 0;
}
```

Wejście: Specyfikacja formatu VI

Dane i wyniki

```
10 20 30 40
y= 20, n= 6
x= 1

1␣␣␣␣␣2222␣ala␣ma␣kota
y=␣2222,␣n=␣11
x=␣1
```

Wejście: Specyfikacja formatu VII

Kolejny przykład

```
#include <stdio.h>
int main(void)
{
    int x, y, z, n;
    x = scanf("%4d␣%5d␣%n", &y, &z, &n);
    printf("y=␣%d,␣z=␣%d,␣n=␣%d\n", y, z, n);
    printf("x=␣%d\n", x);
    return 0;
}
```

Wejście: Specyfikacja formatu VIII

Dane i wyniki

```
1234567890
y= 1234, z= 56789, n= 9
x= 2
```

Wejście: Specyfikacja formatu IX

I jeszcze jeden przykład

```
#include <stdio.h>
int main(void)
{
    int x, y, z, n;
    x = scanf("%4d a %5d %n", &y, &z, &n);
    printf("y= %d, z= %d, n= %d\n", y, z, n);
    printf("x= %d\n", x);
    return 0;
}
```

Wejście: Specyfikacja formatu X

Dane i wyniki

```
123b123b
y=␣123,␣z=␣32767,␣n=␣2023244192
x=␣1

1a2␣3␣4␣5␣6␣7␣8␣9
y=␣1,␣z=␣2,␣n=␣4
x=␣2
```

Politechnika Wrocławska

Wejście: Specyfikacja formatu XI

...

```
#include <stdio.h>
int main(void)
{
    int x, y, z, n;
    x = scanf("%4da%5d%n", &y, &z, &n);
    printf("y= %d, z= %d, n= %d\n", y, z, n);
    printf("x= %d\n", x);
    return 0;
}
```

Wejście: Specyfikacja formatu XII

```
123a123
y= 123, z= 123, n= 7
x= 2
```

Podchwytliwe pytania I

P: Jak powinna wyglądać specyfikacja wprowadzania pozwalająca poprawnie zinterpretować dane w postaci:

`123ala456`

O: Oczywiście tak: %iala%i lub lepiej %dala%d
P: A jak powinna wyglądać specyfikacja pozwalająca przeczytać dane postaci

`123ala456`

oraz

`123ola456`

Odpowiedź będzie dłuższa...

Podchwytliwe pytania II

Popatrzmy na program

```c
#include <stdio.h>
int main(void){
    int x, y, z, n;
    char tekst[100];
    x = scanf("%d%3[a-z]%d%n", &y, tekst, &z, &n);
    printf("y=␣%d␣", y);
    printf("tekst=␣%s␣", tekst);
    printf("z=␣%d␣", z);
    printf("n=␣%d\n", n);
    printf("x=␣%d\n", x);
    return 0;
}
```

Podchwytliwe pytania III

```
123ala20
y= 123 tekst= ala z= 20 n= 8
x= 3

12ooo3456
y= 12 tekst= ooo z= 3456 n= 9
x= 3

123ALA34
y= 123 tekst=  z= 1250361488 n= 0
x= 1
```

Politechnika Wrocławska

Podchwytliwe pytania IV

Jeżeli specyfikację zmienimy na: "%d%*3[a–z]%d%n" to dane tekstowe będą ignorowane, odpowiednie polecenie będzie wyglądać tak:

```
x = scanf("%d%*3[a–z]%d%n", &y, &z, &n);
```

Politechnika Wrocławska

Podchwytliwe pytania V

Co jeszcze może pojawić się wewnątrz nawiasów kwadratowych?

- ciąg znaków — dotyczy wymienionych znaków: [abc]
- znak ^ na pierwszym miejscu — wszystkie znaki za wyjątkiem wymienionych: [^ABC]
- *początek*−*koniec* — znaki z podanego zakresu: [a−z]

Warto też poczytać o wyrażeniach regularnych czasami nazywanych „regułowymi”...

Podchwytliwe pytania VI

P: W jaki sposób przeczytać dowolny napis?

- Jak wiadomo specyfikacja %s czyta znaki do pierwszego odstępu. Zatem nie można w ten sposób przeczytać napisau „Ala ma kota" (ze względu na odstępy.
- Można wspomóc się wyrażeniami regularnymi: specyfikacja %[aA]s czytać będzie dane tekstowe **zgodne z wzorcem**. Czyli tylko litery a lub A.
- Użycie specyfikacji %[^]s powoduje przeczytanie dowolnych znaków aż do znaku odstępu (w istocie znaczy to samo co poprzednio: wzorcem jest dowolny znak różny od spacji!).
- Co spowoduje zatem specyfikacja %[^\n]s?

Rodzina poleceń fprintf I

Wszystkie funkcje realizują formatowane wyprowadzanie informacji.

```
#include <stdarg.h>
#include <stdio.h>
int fprintf(FILE *stream, const char *format, \
            ...);
int printf(const char *format, ...);
int sprintf(char *s, const char *format, ...);
int snprintf(char *str, size_t size, const char *format, ...);
int vfprintf(FILE *stream, const char *format, \
             va_list arg);
int vprintf(const char *format, va_list arg);
int vsprintf(char *s, const char *format, \
             va_list arg);
```

- **fprintf** to podstawowa wersja funkcji

Politechnika Wrocławska

Rodzina poleceń fprintf II

- **printf** używa stdout jako strumienia wyjściowego.
- **sprintf** przekazuje sformatowane informacje do tablicy znakowej (odpowiedniej długości — musi zadbać programista)
- **snprintf** dodatkowy parametr mówi o długości tablicy znakowej, do której mają być zapisywane dane; dłuższe zostaną przycięte!
- warianty o nazwie rozpoczynającej się na v (vfprintf, fprintf, vsprintf) używają specyficznej metody przekazywania listy parametrów zmiennej długości — nie będziemy się nimi zajmować.

Wyjście: Specyfikacje formatu I

Podobnie jak w przypadku formatowanego wprowadzania danych, zmienna lub stała tekstowa zawiera informacje o sposobie wyprowadzania danych. Zasadą jest, że znak % rozpoczyna specyfikację konwersji, a pozostałe znaki są wyprowadzane w postaci takiej jak w formacie.
Specjalne wskaźniki modyfikujące sposób konwersji to:

- `-` wynik będzie justowany do lewej strony (standardowo do prawej)
- `+` Liczby będą zapisywane zawsze ze znakiem
- odstęp odstęp będzie zawsze dodawany przed liczbą (chyba, że liczba poprzedzona jest znakiem)
- `#` Wynik jest konwertowany do postaci „alternatywnej" (zależnej od typu konwersji)

Wyjście: Specyfikacje formatu II

- dla formatu **o** powoduje to zwiększenie precyzji, jeżeli jest to konieczne, aby na początku wyniku było zero;
- dla formatów **x** i **X** niezerowa liczba poprzedzona jest ciągiem 0x lub 0X;
- dla formatów **a**, **A**, **e**, **E**, **f**, **F**, **g** i **G** wynik zawsze zawiera kropkę nawet jeżeli nie ma za nią żadnych cyfr;
- dla formatów **g** i **G** końcowe zera nie są usuwane.

0 Liczby będą poprzedzone wiodącymi zerami (dla formatów **d**, **i**, **o**, **u**, **x**, **X**, **a**, **A**, **e**, **E**, **f**, **F**, **g** i **G**)

Szerokość pola i precyzja I

Minimalna szerokość pola oznacza ile najmniej znaków ma zająć dane pole. Jeżeli wartość po formatowaniu zajmuje mniej miejsca jest ona wyrównywana spacjami z lewej strony (chyba, że podano flagi, które modyfikują to zachowanie). Domyślna wartość tego pola to 0.
Precyzja dla formatów:

- **d**, **i**, **o**, **u**, **x** i **X** określa minimalną liczbę cyfr, które mają być wyświetlone i ma domyślną wartość 1;
- **a**, **A**, **e**, **E**, **f** i **F** — liczbę cyfr, które mają być wyświetlone po kropce i ma domyślną wartość 6;
- **g** i **G** określa liczbę cyfr znaczących i ma domyślną wartość 1;
- dla formatu **s** — maksymalną liczbę znaków, które mają być wypisane.

Szerokość pola i precyzja II

Szerokość pola może być albo dodatnią liczbą zaczynającą się od cyfry różnej od zera albo gwiazdką. Podobnie precyzja z tą różnicą, że jest jeszcze poprzedzona kropką. Gwiazdka oznacza, że brany jest kolejny z argumentów, który musi być typu **int**. Wartość ujemna przy określeniu szerokości jest traktowana tak jakby podano flagę – (minus).

Rozmiar argumentu I

1. Dla formatów **d** i **i** można użyć jednego ze modyfikator rozmiaru:
   - **hh** — oznacza, że format odnosi się do argumentu typu **signed char**,
   - **h** — oznacza, że format odnosi się do argumentu typu **short**,
   - **l** (*el*) — oznacza, że format odnosi się do argumentu typu **long**,
   - **ll** (**el el**) — oznacza, że format odnosi się do argumentu typu **long long**,
   - **j** — oznacza, że format odnosi się do argumentu typu intmax_t,
   - **z** — oznacza, że że format odnosi się do argumentu typu będącego odpowiednikiem typu size_t ze znakiem,
   - **t** — oznacza, że że format odnosi się do argumentu typu ptrdiff_t.
2. Dla formatów **o**, **u**, **x** i **X** można użyć takich samych modyfikatorów rozmiaru jak dla formatu **d** i oznaczają one, że format odnosi się do argumentu odpowiedniego typu bez znaku.

Rozmiar argumentu II

3. Dla formatu **n** można użyć takich samych modyfikatorów rozmiaru jak dla formatu **d** i oznaczają one, że format odnosi się do argumentu będącego wskaźnikiem na dany typ.
4. Dla formatów **a**, **A**, **e**, **E**, **f**, **F**, **g** i **G** można użyć modyfikatorów rozmiaru **L**, który oznacza, że format odnosi się do argumentu typu **long double**.
5. Dodatkowo, modyfikator **l** (*el*) dla formatu **c** oznacza, że odnosi się on do argumentu typu wint_t, a dla formatu **s**, że odnosi się on do argumenty typu wskaźnik na wchar_t.

Politechnika Wrocławska

Format I

Funkcje z rodziny printf obsługują następujące formaty:

d, i — argument typu **int** jest przedstawiany jako liczba całkowita ze znakiem w postaci [−]ddd.

o, u, x, X — argument typu **unsigned int** jest przedstawiany jako nieujemna liczba całkowita zapisana w systemie oktalnym (**o**), dziesiętnym (**u**) lub heksadecymalnym (**x** i **X**).

f, F — argument typu **double** jest przedstawiany w postaci [−]ddd.ddd.

e, E — argument typu **double** jest reprezentowany w postaci [−]d.ddde+dd, gdzie liczba przed kropką dziesiętną jest różna od zera, jeżeli liczba jest różna od zera, a + oznacza znak wykładnika. Format **E** używa wielkiej litery E zamiast małej.

Politechnika Wrocławska

Format II

g, G — argument typu **double** jest reprezentowany w formacie takim jak **f** lub **e** (odpowiednio **F** lub **E**) zależnie od liczby znaczących cyfr w liczbie oraz określonej precyzji.

a, A — argument typu **double** przedstawiany jest w formacie [−]0xh.hhhp+d czyli analogicznie jak dla **e** i **E**, tyle że liczba zapisana jest w systemie heksadecymalnym.

c — argument typu **int** jest konwertowany do **unsigned char** i wynikowy znak jest wypisywany. Jeżeli podano modyfikator rozmiaru **l** argument typu wint_t konwertowany jest do wielobajtowej sekwencji i wypisywany.

s — argument powinien być typu wskaźnik na **char** (lub wchar_t). Wszystkie znaki z podanej tablicy, aż do, i z wyłączeniem znaku null są wypisywane.

Format III

- p — argument powinien być typu wskaźnik na **void**. Jest to konwertowany na serię drukowalnych znaków w sposób zależny od implementacji.
- n — argument powinien być wskaźnikiem na liczbę całkowitą ze znakiem, do którego zapisana jest liczba zapisanych znaków.

Część II

Binarne wejście/wyjście

Politechnika Wrocławska

„Otwarcie" pliku I

```
#include <stdio.h>
FILE *fopen(const char *filename,
            const char *mode);
FILE *freopen(const char *filename,
              const char *mode, FILE *stream);
...
FILE *fp;
```

- Pierwszy parametr (filename) to wskaźnik do tablicy tekstowej (na przykład stałej) zawierającej nawę pliku.
- Parametr drugi to ciąg znaków (lub zmienna) zawierająca w sposób symboliczny określenie trybu dostępu:
  - **rb** otwórz plik binarny do czytania

Politechnika Wrocławska

„Otwarcie" pliku II

- **wb** utwórz plik do zapisu w trybie binarnym, jeżeli plik istnieje, jego zawartość zostanie skasowana
- **ab** otwórz plik binarny w trybie do dopisywania (jeżeli plik nie istnieje — zostanie utworzony).
- **r+b** lub **rb+** zapis i odczyt dla plików binarnych (plik musi istnieć)
- **w+b** lub **wb+** otwórz w trybie binarnym plik istniejący (kasując jego zawartość) lub utwórz plik nowy w trybie do zapisu i odczytu
- **a+b** lub **ab+** otwórz plik w trybie binarnym na końcu w trybie do zapisu i odczytu

- Trzeci parametr (stream — tylko w przypadku funkcji reopen) zawiera wskaźnik do struktury danych opisujących strumień danych.

Politechnika Wrocławska

Zapis/Odczyt binarny I

1. **fread()**

```
#include <stdio.h>
size_t fread(void *ptr, size_t size,
             size_t nmemb, FILE *stream);
```

Funkcja realizująca dostęp bezpośredni do pliku, pozwala odczytać **nmemb** elementów o wielkości **size** każdy do tablicy wskazywanej przez **ptr** ze strumienia **stream**.
Funkcja zwraca liczbę przeczytanych elementów, która może być mniejsza od **nmemb** gdy wystąpi błąd lub funkcja dojdzie do końca pliku.

Zapis/Odczyt binarny II

2. **fwrite()**

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t size,
              size_t nmemb, FILE *stream);
```

Funkcja zapisuje binarnie elementy tablicy wskazanej przez **ptr** do strumienia stream. **size** określa wielkość jednego obiektu, a **nmemb** liczbę obiektów.

Zapis tablicy I

```
#include <stdio.h>
int main()
{
    FILE *pp;
    double t[10] = {
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9
    };
    pp = fopen("Ala.txt", "wb");
    if ( pp == NULL )
    {
        printf("Blad!\n");
        return 1;
    }
```

Zapis tablicy II

```
    fwrite(t, sizeof( double ), 10, pp);
    fclose(pp);
    return 0;
}
```

Politechnika Wrocławska

Pliki binarne I

Prosty przykład

```c
/*
 * Odczyt jednej liczby w trybie binarnym
 */
#include <stdio.h>
int main(void)
{
   FILE *fp;
   unsigned int e;
   fp = fopen("foo.txt", "rb");
   if ( fp == NULL )
   {
      printf("blad1\n");
      return 1;
```

Pliki binarne II

Prosty przykład

```
    }
    if ( fread(&e, sizeof( e ), 1, fp) == 0 )
    {
        printf("blad2\n");
        return 1;
    }
    printf("e = %u\n", e);
    return 0;
}
```

Pliki binarne III

Prosty przykład

1. Plik foo.txt nie istnieje

```
$ ls
b.c  Debug
$ Debug/binarne
blad1
```

Pliki binarne IV

Prosty przykład

2. Plik foo.txt istnieje, ale jest pusty

```
$ touch foo.txt
$ ls -l
razem 8
-rw-r--r-- 1 myszka myszka  319 2008-05-05 12:05 b.c
drwxr-xr-x 2 myszka myszka 4096 2008-05-05 12:05 Debug
-rw-r--r-- 1 myszka myszka    0 2008-05-05 12:12 foo.txt
$ Debug/binarne
blad2
```

Pliki binarne V

Prosty przykład

3. Plik istnieje, ma długość trzech bajtów:

```
$ ls -l
razem 12
-rw-r--r-- 1 myszka myszka  319 2008-05-05 12:05 b.c
drwxr-xr-x 2 myszka myszka 4096 2008-05-05 12:05 Debug
-rw-r--r-- 1 myszka myszka    3 2008-05-05 12:15 foo.txt
$ Debug/binarne
blad2
```

Pliki binarne VI

Prosty przykład

4. Plik istnieje, ma długość czterech bajtów:

```
$ cat foo.txt
abc
$ ls -l
razem 12
-rw-r--r-- 1 myszka myszka  319 2008-05-05 12:05 b.c
drwxr-xr-x 2 myszka myszka 4096 2008-05-05 12:05 Debug
-rw-r--r-- 1 myszka myszka    4 2008-05-05 12:18 foo.txt
$ Debug/binarne
e = 174285409
```

Politechnika Wrocławska

Zapis i odczyt I

```
#include <stdio.h>
int main(void)
{
    FILE *fp;
    unsigned int i, k;
    fp = fopen("foo.txt", "wb");
    if ( fp == NULL )
    {
        printf("blad1\n");
        return 1;
    }
    for ( i = 0; i < 10; i++ )
    {
```

Zapis i odczyt II

```
        fseek(fp, i * sizeof( i ), SEEK_SET);
        fwrite(&i, sizeof( i ), 1, fp);
    }
    fclose(fp);
    fp = fopen("foo.txt", "rb");
    if ( fp == NULL )
    {
        printf("blad1\n");
        return 1;
    }
    for ( i = 0; i < 10; i++ )
    {
        fseek(fp, ( 9 - i ) * sizeof( i ), SEEK_SET);
        fread(&k, sizeof( i ), 1, fp);
```

Zapis i odczyt III

```
        printf("k=␣%d\n", k);
    }
    fclose(fp);
    return 0;
}
```

Zapis i odczyt (wariant drugi) I

```c
#include <stdio.h>
int main(void)
{
    FILE *fp;
    unsigned int i, k;
    fp = fopen("foo.txt", "w+b");
    if ( fp == NULL )
    {
        printf("blad1\n");
        return 1;
    }
    for ( i = 0; i < 10; i++ )
    {
```

Zapis i odczyt (wariant drugi) II

```
        fseek(fp, i * sizeof( i ), SEEK_SET);
        fwrite(&i, sizeof( i ), 1, fp);
    }
    for ( i = 0; i < 10; i++ )
    {
        fseek(fp, ( 9 - i ) * sizeof( i ), SEEK_SET);
        fread(&k, sizeof( i ), 1, fp);
        printf("k= %d\n", k);
    }
    fclose(fp);
    return 0;
}
```