

# Informatyka 2

Politechnika Białostocka - Wydział Elektryczny  
Elektrotechnika, semestr III, studia stacjonarne I stopnia  
Rok akademicki 2017/2018

## Wykład nr 4 (06.11.2017)

dr inż. Jarosław Forenc

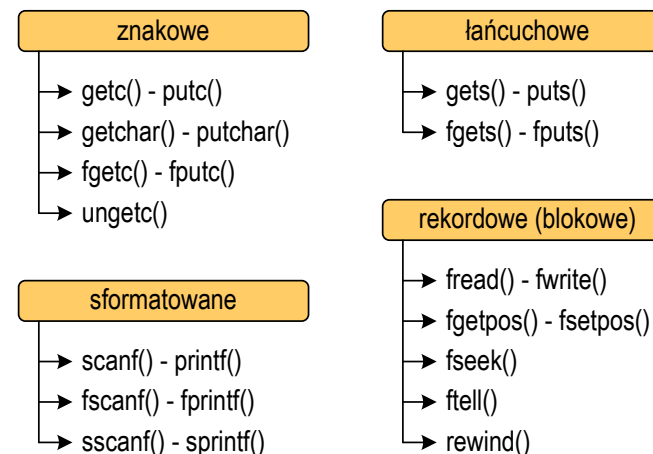
## Plan wykładu nr 4

- Operacje wejścia-wyjścia w języku C
- Strumienie
- Operacje na plikach
  - otwarcie pliku
  - zamknięcie pliku
- Typy operacji wejścia-wyjścia
  - znakowe
  - łańcuchowe
  - sformatowane
  - rekordowe (blokowe)

## Operacje wejścia-wyjścia w języku C

- Operacje wejścia-wyjścia nie są elementami języka C
- Zostały zrealizowane jako funkcje zewnętrzne, znajdujące się w bibliotekach dostarczanych wraz z kompilatorem
- **Standardowe** wejście-wyjście (strumieniowe)
  - plik nagłówkowy `stdio.h`
  - duża liczba funkcji, proste w użyciu
  - ukrywa przed programistą szczegóły wykonywanych operacji
- **Systemowe** wejście-wyjście (deskryptorowe, niskopoziomowe)
  - plik nagłówkowy `io.h`
  - mniejsza liczba funkcji
  - programista sam obsługuje szczegóły wykonywanych operacji
  - funkcje bardziej zbliżone do systemu operacyjnego - działają szybciej

## Typy standardowych operacji wejścia-wyjścia



## Strumienie

- Standardowe operacje wejścia-wyjścia opierają się na **strumieniach** (ang. **stream**)
- Strumień jest pojęciem abstrakcyjnym - jego nazwa bierze się z analogii między przepływem danych, a np. wody
- W strumieniu dane płyną od źródła do odbiorcy
- Użytkownik określa źródło i odbiorcę, typ danych oraz sposób ich przesyłania
- Strumień może być skojarzony ze zbiorem danych znajdujących się na dysku (plik) lub zbiorem danych pochodzących z urządzenia znakowego (klawiatura)
- Niezależnie od fizycznego medium, z którym strumień jest skojarzony, wszystkie strumienie mają podobne właściwości

## Strumienie

- Strumienie reprezentowane są przez zmienne będące wskaźnikami na struktury typu **FILE** (definicja w pliku **stdio.h**)

```
struct _iobuf {
    char *_ptr;
    int _cnt;
    char *_base;
    int _flag;
    int _file;
    int _charbuf;
    int _bufsiz;
    char *_tmpfname;
};
typedef struct _iobuf FILE;
```

- Podczas pisania programów nie ma potrzeby bezpośredniego odwoływania się do pól tej struktury

## Strumienie

- W każdym programie automatycznie tworzone są i otwierane trzy standardowe strumienie wejścia-wyjścia:
  - **stdin** - standardowe wejście, skojarzone z klawiaturą
  - **stdout** - standardowe wyjście, skojarzone z ekranem monitora
  - **stderr** - standardowe wyjście dla komunikatów o błędach, skojarzone z ekranem monitora

```
_CRTIMP FILE * __cdecl __iob_func(void);
#define stdin (&__iob_func()[0])
#define stdout (&__iob_func()[1])
#define stderr (&__iob_func()[2])
```

- Funkcja **printf()** niejawnie używa strumienia **stdout**
- Funkcja **scanf()** niejawnie używa strumienia **stdin**

## Operacje na plikach

- Strumień wiąże się z plikiem za pomocą **otwarcia**, zaś połączenie to jest przerywane przez **zamknięcie** strumienia
- Operacje związane z przetwarzaniem pliku zazwyczaj składają się z trzech części

1. Otwarcie pliku (strumienia):  
- funkcje: **fopen()**

2. Operacje na pliku (strumieniu), np. czytanie, pisanie:  
- funkcje dla plików tekstowych: **fprintf()**, **fscanf()**, **fgetc()**,  
**fputc()**, **fgets()**, **fputs()**...  
- funkcje dla plików binarnych: **fread()**, **fwrite()**, ...

3. Zamknięcie pliku (strumienia):  
- funkcja: **fclose()**

## Otwarcie pliku - fopen()

```
FOPEN stdio.h  
FILE* fopen(const char *fname, const char *mode);
```

- Otwiera plik o nazwie **fname**, nazwa może zawierać całą ścieżkę dostępu do pliku
- **mode** określa tryb otwarcia pliku:
  - "r" - odczyt
  - "w" - zapis - jeśli pliku nie ma to zostanie on utworzony, jeśli plik istnieje, to jego poprzednia zawartość zostanie usunięta
  - "a" - zapis (dopisywanie) - dopisywanie danych na końcu istniejącego pliku, jeśli pliku nie ma to zostanie utworzony

## Otwarcie pliku - fopen()

```
FOPEN stdio.h  
FILE* fopen(const char *fname, const char *mode);
```

- Otwiera plik o nazwie **fname**, nazwa może zawierać całą ścieżkę dostępu do pliku
- **mode** określa tryb otwarcia pliku:
  - "r+" - uaktualnienie (zapis i odczyt)
  - "w+" - uaktualnienie (zapis i odczyt) - jeśli pliku nie ma to zostanie on utworzony, jeśli plik istnieje, to jego poprzednia zawartość zostanie usunięta
  - "a+" - uaktualnienie (zapis i odczyt) - dopisywanie danych na końcu istniejącego pliku, jeśli pliku nie ma to zostanie utworzony, odczyt może dotyczyć całego pliku, zaś zapis może polegać tylko na dodawaniu nowych danych

## Otwarcie pliku - fopen()

```
FOPEN stdio.h  
FILE* fopen(const char *fname, const char *mode);
```

- Zwraca wskaźnik na strukturę **FILE** skojarzoną z otwartym plikiem
- Gdy otwarcie pliku nie powiodło się to zwraca **NULL**
- Zawsze należy sprawdzać, czy otwarcie pliku powiodło się
- Po otwarciu pliku odwołujemy się do niego przez wskaźnik pliku
- Domyślnie plik jest otwierany w **trybie tekstowym**, natomiast dodanie litery "b" w trybie otwarcie oznacza **tryb binarny**

## Otwarcie pliku - fopen()

- Otwarcie pliku w trybie tekstowym, tylko odczyt

```
FILE *fp;  
fp = fopen("dane.txt", "r");
```

- Otwarcie pliku w trybie binarnym, tylko zapis

```
fp = fopen("c:\\baza\\data.bin", "wb");
```

- Otwarcie pliku w trybie tekstowym, tylko zapis

```
fp = fopen("wynik.txt", "wt");
```



## Format (plik) tekstowy i binarny

- Dane w pliku tekstowym zapisane są w postaci kodów ASCII
- Deklaracja i inicjalizacja zmiennej `x` typu `int`:

```
int x = 123456;
```

- W pamięci komputera zmienna `x` zajmuje 4 bajty:

```
00000000 00000001 11100010 01000000 (2)
```

- Po zapisaniu wartości zmiennej `x` do pliku **binarnego** znajdują się w nim 4 bajty o takiej samej zawartości jak w pamięci komputera

```
00000000 00000001 11100010 01000000 (2)
```

## Format (plik) tekstowy i binarny

- Elementami pliku tekstowego są **wiersze** o różnej długości
- W systemach DOS/Windows każdy wiersz pliku tekstowego zakończony jest parą znaków:
  - CR (carriage return) - powrót karetki, kod ASCII -  $13_{(10)} = 0D_{(16)} = \backslashr'$
  - LF (line feed) - przesunięcie o wiersz, kod ASCII -  $10_{(10)} = 0A_{(16)} = \backslashn'$
- Załóżmy, że plik tekstowy ma postać:

```
Pierwszy wiersz pliku
Drugi wiersz pliku
Trzeci wiersz pliku
```

- Rzeczywista zawartość pliku jest następująca:

```
50 69 65 72 77 73 7A 79|20 77 69 65 72 73 7A 20 | Pierwszy wiersz
70 6C 69 6B 75 0D 0A|44 72 75 67 69 20 77 69 65 | plikuDrugi wie
72 73 7A 20 70 6C 69 6B|75 0D 0A 54 72 7A 65 63 | rsz plikuTrzec
69 20 77 69 65 72 73 7A|20 70 6C 69 6B 75 0D 0A | i wiersz pliku
```

## Format (plik) tekstowy i binarny

- W systemie Linux każdy wiersz pliku tekstowego zakończony jest tylko jednym znakiem:
  - LF (line feed) - przesunięcie o wiersz, kod ASCII -  $10_{(10)} = 0A_{(16)} = \backslashn'$
- Załóżmy, że plik tekstowy ma postać:

```
Pierwszy wiersz pliku
Drugi wiersz pliku
Trzeci wiersz pliku
```

- Rzeczywista zawartość pliku jest następująca:

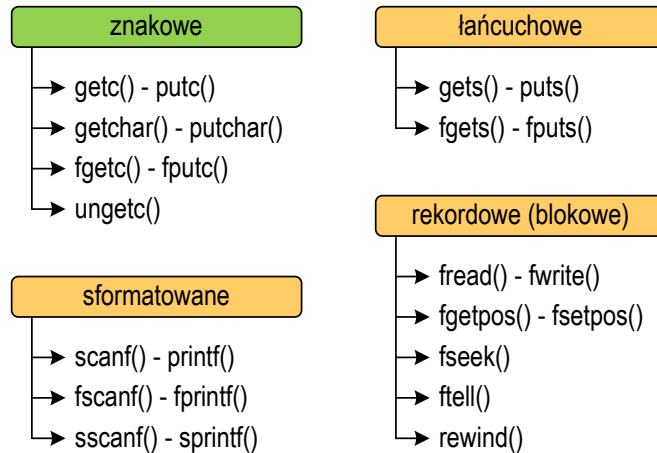
```
50 69 65 72 77 73 7A 79|20 77 69 65 72 73 7A 20 | Pierwszy wiersz
70 6C 69 6B 75 0A|44 72 75 67 69 20 77 69 65 72 | plikuDrugi wier
73 7A 20 70 6C 69 6B 75|0A 54 72 7A 65 63 69 20 | sz plikuTrzeci
77 69 65 72 73 7A 20 70|6C 69 6B 75 0A | wiersz pliku
```

- Pliki **binarne** nie mają ściśle określonej struktury

## Tryby otwarcia pliku: tekstowy i binarny

- Różnice pomiędzy trybem tekstowym i binarnym otwarcia pliku dotyczą innego traktowania znaków CR i LF
- W trybie **tekstowym**:
  - przy odczycie pliku para znaków CR, LF jest tłumaczona na znak nowej linii (LF)
  - przy zapisie pliku znak nowej linii (LF) jest zapisywany w postaci dwóch znaków (CR, LF)
- W trybie **binarnym**:
  - przy odczycie i zapisie para znaków CR, LF jest traktowana zawsze jako dwa znaki

## Znakowe operacje wejścia-wyjścia



## Znakowe operacje wejścia-wyjścia

```
GETC stdio.h  
int getc(FILE *fp);
```

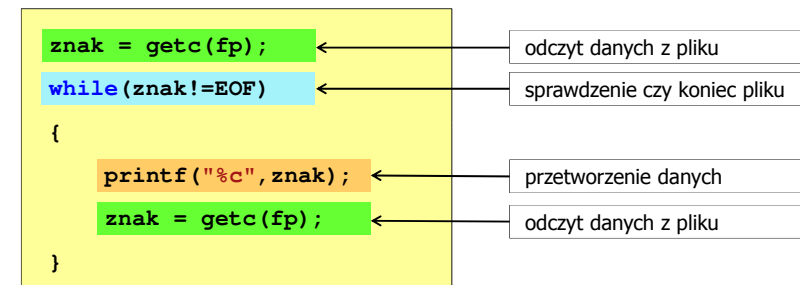
- Pobiera jeden znak z aktualnej pozycji otwartego strumienia `fp` i uaktualnia pozycję
- Zmienna `fp` powinna wskazywać strukturę `FILE` reprezentującą strumień skojarzony z otwartym plikiem lub jeden ze standardowo otwartych strumieni (np. `stdin`)
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca wartość całkowitą `ko`du wczytanego znaku (typ `int`)
- Jeśli wystąpił błąd lub przeczytany został znacznik końca pliku, to funkcja zwraca wartość `EOF`

## Przykład: wyświetlenie pliku tekstowego

```
#include <stdio.h>  
int main(void)  
{  
    FILE *fp;  
    int znak;  
  
    fp = fopen("test.txt", "r");  
    znak = getc(fp);  
    while(znak!=EOF)  
    {  
        printf("%c", znak);  
        znak = getc(fp);  
    }  
  
    fclose(fp);  
    return 0;  
}
```

## Schemat przetwarzania pliku

- Typowy schemat odczytywania danych z pliku



## Przykład: wyświetlenie pliku tekstowego

- Odczytanie i wyświetlenie zawartości pliku tekstowego

```
znak =getc(fp);  
while(znak!=EOF)  
{  
    printf("%c",znak);  
    znak =getc(fp);  
}
```

można zapisać w krótszej postaci:

```
while((znak=getc(fp))!=EOF)  
    printf("%c",znak);
```

## Znakowe operacje wejścia-wyjścia

PUTC

stdio.h

```
int putc(int znak, FILE *fp);
```

- Wpisuje **znak** do otwartego strumienia reprezentowanego przez argument **fp**
- Zmienna **fp** powinna wskazywać strukturę **FILE** reprezentującą strumień skojarzony z otwartym plikiem lub jeden ze standardowo otwartych strumieni (np. **stdout**)
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca wypisany **znak**
- Jeśli wystąpił błąd, to funkcja zwraca wartość **EOF**

## Przykład: zapisanie alfabetu do pliku tekstowego

```
#include <stdio.h>  
int main(void)  
{  
    FILE *fp = fopen("alfabet.txt", "w");  
    for (int i='A'; i<='Z'; i++)  
        putc(i, fp);  
    fclose(fp);  
    return 0;  
}
```

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- Stosując strumień **stdout** można wyświetlić alfabet na ekranie

```
for (int i='A'; i<='Z'; i++)  
    putc(i, stdout);
```

## Znakowe operacje wejścia-wyjścia

GETCHAR

stdio.h

```
int getchar(void);
```

- Pobiera znak ze strumienia **stdin** (klawiatura)
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca przeczytany znak (typ **int**)
- Jeśli wystąpił błąd albo został przeczytany znacznik końca pliku, to funkcja zwraca wartość **EOF**

```
int znak;  
znak = getchar();  
printf("%c", znak);
```

## Znakowe operacje wejścia-wyjścia

**PUTCHAR** stdio.h  
`int putchar(int znak);`

- Wpisuje **znak** do strumienia **stdout** (standardowo ekran)
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca wypisany **znak**
- Jeśli wystąpił błąd, to funkcja zwraca wartość **EOF**

```
int znak = 'x';  
putchar(znak);
```

## Przykład: liczba znaków wczytanych z klawiatury

```
#include <stdio.h>  
  
int main(void)  
{  
    int znak, ile = 0;  
    while ((znak=getchar())!='\n')  
        ile++;  
    printf("Liczba znakow: %d\n",ile);  
    return 0;  
}
```

```
Ala ma laptopa  
Liczba znakow: 14
```

- Wprowadzane znaki są buforowane do naciśnięcia klawisza **Enter**
- Po naciśnięciu klawisza **Enter** zawartość bufora jest przesyłana do programu i analizowana w nim

## Znakowe operacje wejścia-wyjścia

**FGETC** stdio.h  
`int fgetc(FILE *fp);`

- Pobiera jeden znak ze strumienia wskazywanego przez **fp**
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca przeczytany znak po przekształceniu go na typ **int**
- Jeśli wystąpił błąd lub został przeczytany znacznik końca pliku, to funkcja zwraca wartość **EOF**

## Znakowe operacje wejścia-wyjścia

**FPUTC** stdio.h  
`int fputc(int znak, FILE *fp);`

- Wpisuje **znak** do otwartego strumienia reprezentowanego przez argument **fp**
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca wypisany **znak** (typ **int**)
- Jeśli wystąpił błąd, to funkcja zwraca wartość **EOF**



## Przykład: liczba wyrazów w pliku

```
#include <stdio.h>

int main(void)
{
    FILE *fp;
    int znak, odstep = 1, ile = 0;

    fp = fopen("test.txt", "r");
    while ((znak = fgetc(fp)) != EOF)
        if (znak == ' ' || znak == '\t' || znak == '\n')
            odstep = 1;
        else
            if (odstep != 0) { odstep = 0; ile++; }
    fclose(fp);
    printf("Liczba slow: %d\n", ile);

    return 0;
}
```

Ala ma laptopa i psa.

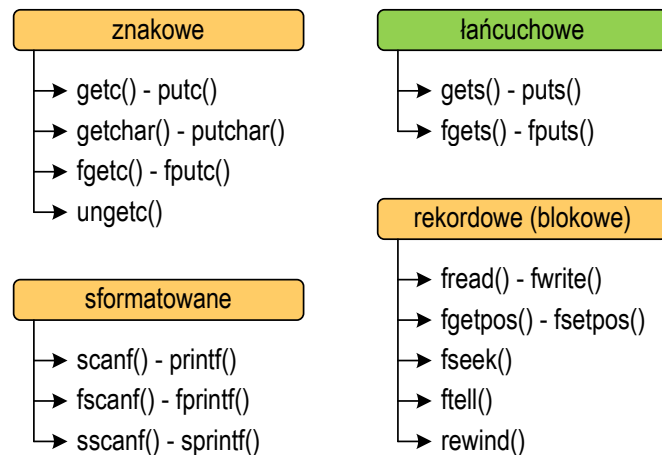
Liczba slow: 5

## Znakowe operacje wejścia-wyjścia

**UNGETC** stdio.h  
`int ungetc(int znak, FILE *fp);`

- Umieszcza **znak** z powrotem w strumieniu wejściowym **fp**

## Łańcuchowe operacje wejścia-wyjścia



## Łańcuchowe operacje wejścia-wyjścia

**GETS** stdio.h  
`char* gets(char *buf);`

- Pobiera do bufora pamięci wskazywanego przez argument **buf** linię znaków ze strumienia **stdin** (standardowo klawiatura)
- Wczytywanie jest kończone po napotkaniu znacznika nowej linii **'\n'**, który zastępowany jest znakiem końca łańcucha **'\0'**
- Funkcja **gets()** umożliwia wczytanie łańcucha znaków zawierającego spacje i tabulatory
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca wskazanie do łańcucha **buf**
- Jeśli wystąpił błąd lub podczas wczytywania został napotkany znacznik końca pliku, to funkcja zwraca wartość **EOF**

## Łańcuchowe operacje wejścia-wyjścia

**PUTS** stdio.h  
`int puts(const char *buf);`

- Wpisuje łańcuch `buf` do strumienia `stdout` (standardowo ekran), zastępując znak `'\0'` znakiem `'\n'`
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca ostatni wypisany znak
- Jeśli wystąpił błąd, to funkcja zwraca wartość `EOF`

```
char tablica[80];  
gets(tablica);  
puts(tablica);
```

## Łańcuchowe operacje wejścia-wyjścia

**FGETS** stdio.h  
`char* fgets(char *buf, int max, FILE *fp);`

- Pobiera znaki z otwartego strumienia reprezentowanego przez `fp` i zapisuje je do bufora pamięci wskazanego przez `buf`
- Pobieranie znaków jest przerywane po napotkaniu znacznika końca linii `'\n'` lub odczytaniu `max-1` znaków
- Po ostatnim przeczytanym znaku wstawia do bufora `buf` znak `'\0'`
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca wskazanie do łańcucha `buf`
- Jeśli wystąpił błąd lub napotkano znacznik końca pliku, to funkcja zwraca wartość `NULL`

## Łańcuchowe operacje wejścia-wyjścia

**FPUTS** stdio.h  
`int fputs(const char *buf, FILE *fp);`

- Wpisuje łańcuch `buf` do strumienia `fp`, nie dołącza znaku końca wiersza `'\n'`
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca ostatni wypisany znak
- Jeśli wystąpił błąd, to funkcja zwraca wartość `EOF`

## Przykład: wyświetlenie pliku tekstowego

```
#include <stdio.h>  
  
int main(void)  
{  
    FILE *fp;  
    char buf[15];  
  
    fp = fopen("test.txt", "r");  
    while (fgets(buf, 15, fp) != NULL)  
        fputs(buf, stdout);  
  
    fclose(fp);  
    return 0;  
}
```

## Przykład: wyświetlenie pliku tekstowego

- Zawartość pliku `test.txt`

```
Poprzednikiem języka CCRLF  
był język B,CRLF  
któryCRLF  
Ritchie rozwinał w język C.CRLF
```

- Kolejne wywołania funkcji `fgets(buf,15,fp);`

```
Poprzednikiem języka CCRLF  
był język B,CRLF  
któryCRLF  
Ritchie rozwinał w język C.CRLF
```

## Przykład: wyświetlenie pliku tekstowego

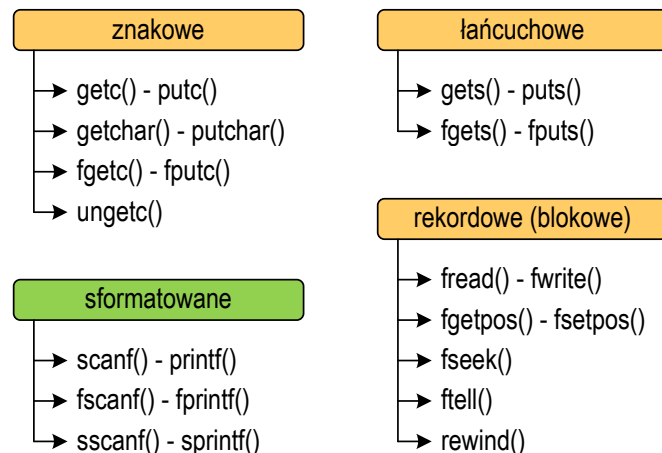
- Kolejne wywołania funkcji `fgets(buf,15,fp);` i zawartość tablicy `buf`

```
Poprzednikiem języka CCRLF  
był język B,CRLF  
któryCRLF  
Ritchie rozwinał w język C.CRLF
```

P	o	p	r	z	e	d	n	i	k	i	e	m	\0
j	e	z	y	k	a	C	\n	\0					
b	y	l	j	e	z	y	k	B	,	\n	\0		
k	t	o	r	y	\n	\0							
R	i	t	c	h	i	e	r	o	z	w	i	n	\0
a	l	w	j	e	z	y	k	C	.	\n	\0		

<sup>LF</sup> = `\n`

## Sformatowane operacje wejścia-wyjścia



## Sformatowane operacje wejścia-wyjścia

- SCANF** `stdio.h`  
`int scanf(const char *format, ...);`
  - Czyta dane ze strumienia `stdin` (klawiatura)
- FSCANF** `stdio.h`  
`int fscanf(FILE *fp, const char *format, ...);`
  - Czyta dane z otwartego strumienia (pliku) `fp`
- SSCANF** `stdio.h`  
`int sscanf(char *buf, const char *format, ...);`
  - Czyta dane z bufora pamięci wskazywanego przez `buf`

## Sformatowane operacje wejścia-wyjścia

```
PRINTF stdio.h  
int printf(const char *format, ...);
```

- Wyprowadza dane do strumienia `stdout` (ekran)

```
FPRINTF stdio.h  
int fprintf(FILE *fp, const char *format, ...);
```

- Wyprowadza dane do otwartego strumienia (pliku) `fp`

```
SPRINTF stdio.h  
int sprintf(char *buf, const char *format, ...);
```

- Wyprowadza dane do bufora pamięci wskazywanego przez `buf`

## Przykład: zapisanie danych do pliku tekstowego

```
#include <stdio.h>  
  
int main(void)  
{  
    FILE *fp;  
    int wiek = 21;  
    float wzrost = 1.78f;  
    char imie[10] = "Jan", nazw[10] = "Kowalski";  
  
    fp = fopen("dane.txt", "w");  
    fprintf(fp, "Imie: %s\n", imie);  
    fprintf(fp, "Nazwisko: %s\n", nazw);  
    fprintf(fp, "Wiek: %d [lat]\n", wiek);  
    fprintf(fp, "Wzrost: %.2f [m]\n", wzrost);  
    fclose(fp);  
  
    return 0;  
}
```

```
Imie:      Jan  
Nazwisko:  Kowalski  
Wiek:      21 [lat]  
Wzrost:    1.78 [m]
```

## Obsługa błędów wejścia-wyjścia

```
FEOF stdio.h  
int feof(FILE *fp);
```

- Sprawdza, czy podczas ostatniej operacji wejścia dotyczącej strumienia `fp` został osiągnięty koniec pliku
- Zwraca wartość różną od zera, jeśli podczas ostatniej operacji wejścia został wykryty koniec pliku, w przeciwnym razie zwraca wartość `0` (zero)

## Przykład: odczytanie liczb z pliku tekstowego

```
#include <stdio.h>  
  
int main(void)  
{  
    FILE *fp; int x;  
    fp = fopen("liczby.txt", "r");  
    fscanf(fp, "%d", &x);  
    while(!feof(fp))  
    {  
        printf("%d\n", x);  
        fscanf(fp, "%d", &x);  
    }  
    fclose(fp);  
    return 0;  
}
```

```
3  
51  
93  
10  
93  
64  
63  
0  
46  
24
```

## Przykład: odczytanie liczb z pliku tekstowego

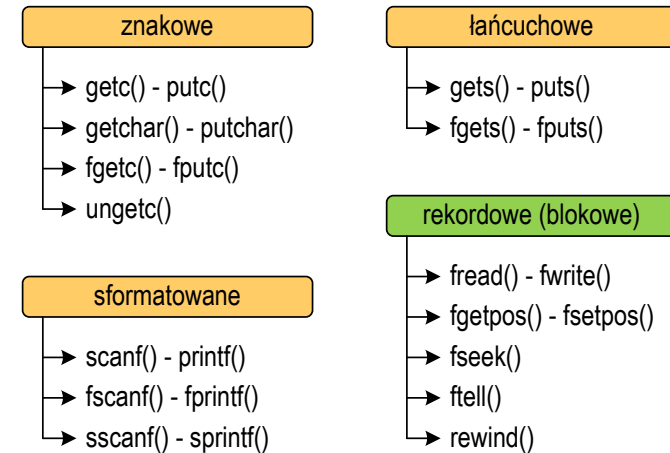
- Sposób zapisu liczb w pliku wejściowym nie ma znaczenia dla prawidłowości ich odczytu
- Liczby powinny być oddzielone od siebie znakami spacji, tabulacji lub znakiem nowego wiersza

```
3 51 93 10 93 64 63 0 46 24
```

```
3 51 93 10  
93 64 63 0  
46 24
```

```
3  
51  
93  
10  
93  
64  
63  
0  
46  
24
```

## Rekordowe (blokowe) operacje wejścia-wyjścia



## Rekordowe (blokowe) operacje wejścia-wyjścia

```
FWRITE stdio.h  
size_t fwrite(const void *p, size_t s, size_t n,  
FILE *fp);
```

- Zapisuje **n** elementów o rozmiarze **s** bajtów każdy, do pliku wskazywanego przez **fp**, biorąc dane z obszaru pamięci wskazywanego przez **p**
- Zwraca liczbę zapisanych elementów - jeśli jest ona różna od **n**, to wystąpił błąd zapisu (brak miejsca na dysku lub dysk zabezpieczony przed zapisem)

## Przykład: zapisanie danych do pliku binarnego

```
#include <stdio.h>

int main(void)
{
    FILE *fp;
    int x = 10, tab[5] = {1,2,3,4,5};
    float y = 1.2345f;

    fp = fopen("dane.dat", "wb");
    fwrite(&x, sizeof(int), 1, fp);
    fwrite(tab, sizeof(int), 5, fp);
    fwrite(tab, sizeof(tab), 1, fp);
    fwrite(&y, sizeof(float), 1, fp);
    fclose(fp);

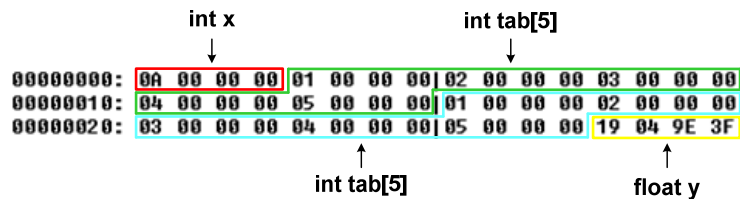
    return 0;
}
```

## Przykład: zapisanie danych do pliku binarnego

- Czterokrotne wywołanie funkcji `fwrite()`

```
fwrite (&x, sizeof(int), 1, fp); // int x = 10;
fwrite (tab, sizeof(int), 5, fp); // int tab[5] = {1,2,3,4,5};
fwrite (tab, sizeof(tab), 1, fp); // int tab[5] = {1,2,3,4,5};
fwrite (&y, sizeof(float), 1, fp); // float y = 1.2345;
```

spowoduje zapisanie do pliku 48 bajtów:



## Rekordowe (blokowe) operacje wejścia-wyjścia

**FREAD**

**stdio.h**

```
size_t fread(void *p, size_t s, size_t n,
             FILE *fp);
```

- Pobiera `n` elementów o rozmiarze `s` bajtów każdy, z pliku wskazywanego przez `fp` i umieszcza odczytane dane w obszarze pamięci wskazywanym przez `p`
- Zwraca liczbę odczytanych elementów - w przypadku gdy liczba ta jest różna od `n`, to wystąpił błąd końca strumienia (w pliku było mniej elementów niż podana wartość argumentu `n`)

## Przykład: odczytanie liczb z pliku binarnego

```
#include <stdio.h>

int main(void)
{
    FILE *fp;
    int x, ile = 0;

    fp = fopen("liczby.dat", "rb");
    fread(&x, sizeof(int), 1, fp);
    while (!feof(fp))
    {
        ile++; printf("%d\n", x);
        fread(&x, sizeof(int), 1, fp);
    }
    fclose(fp);
    printf("Odczytano: %d liczb\n", ile);
    return 0;
}
```

```
37
31
83
27
6
62
31
50
Odczytano: 8 liczb
```

## Przykład: odczytanie liczb z pliku binarnego

- Po otwarciu pliku wskaźnik pozycji pliku pokazuje na jego początek

```
↓
25 00 00 00 1F 00 00 00|53 00 00 00 1B 00 00 00 | %■■■■■■■■S■■■■■■■■
06 00 00 00 3E 00 00 00|1F 00 00 00 32 00 00 00 | ■■■■>■■■■■■■■2■■■
```

- Po odczytaniu jednej liczby: `fread(&x,sizeof(int),1,plik);`  
wskaźnik jest automatycznie przesuwany o `sizeof(int)` bajtów

```
↓
25 00 00 00 1F 00 00 00|53 00 00 00 1B 00 00 00 | %■■■■■■■■S■■■■■■■■
06 00 00 00 3E 00 00 00|1F 00 00 00 32 00 00 00 | ■■■■>■■■■■■■■2■■■
```

- Po odczytaniu kolejnej liczby: `fread(&x,sizeof(int),1,plik);`  
wskaźnik jest ponownie przesuwany o `sizeof(int)` bajtów

```
↓
25 00 00 00 1F 00 00 00|53 00 00 00 1B 00 00 00 | %■■■■■■■■S■■■■■■■■
06 00 00 00 3E 00 00 00|1F 00 00 00 32 00 00 00 | ■■■■>■■■■■■■■2■■■
```

- Plik binarny zawiera liczby: 37 31 83 27 6 62 31 50

## Rekordowe (blokowe) operacje wejścia-wyjścia

```
REWIND stdio.h  
void rewind(FILE *fp);
```

- Ustawia wskaźnik pozycji w pliku wskazywanym przez `fp` na początek pliku

```
FTELL stdio.h  
long int ftell(FILE *fp);
```

- Zwraca bieżące położenie w pliku wskazywanym przez `fp` (liczbę bajtów od początku pliku)

## Przykład: ile razy występuje w pliku wartość max

```
#include <stdio.h>  
  
int main(void)  
{  
    FILE *fp;  
    int x, max, ile = 0;  
  
    fp = fopen("dane.dat", "rb");  
  
    fread(&x, sizeof(int), 1, fp);  
    max = x;  
    while (!feof(fp))  
    {  
        if (x > max) max = x;  
        fread(&x, sizeof(int), 1, fp);  
    }  
    printf("Wartosc max: %d\n", max);  
}
```

```
7 3 3 0 3 9 6 4 1 8  
6 0 4 5 4 9 4 5 4 5  
9 9 8 0 0 5 3 5 1 0
```

## Przykład: ile razy występuje w pliku wartość max

```
rewind(fp);  
  
fread(&x, sizeof(int), 1, fp);  
while (!feof(fp))  
{  
    if (x == max) ile++;  
    fread(&x, sizeof(int), 1, fp);  
}  
printf("Wystapienia max: %d\n", ile);  
  
fclose(fp);  
  
return 0;  
}
```

```
7 3 3 0 3 9 6 4 1 8  
6 0 4 5 4 9 4 5 4 5  
9 9 8 0 0 5 3 5 1 0
```

```
Wartosc max: 9  
Wystapienia max: 4
```

## Rekordowe (blokowe) operacje wejścia-wyjścia

```
FSEEK stdio.h  
int fseek(FILE *fp, long int offset, int mode);
```

- Pozwala przejść bezpośrednio do dowolnego bajtu w pliku wskazywanym przez `fp`
- `offset` określa wielkość przejścia w bajtach, zaś `mode` - punkt początkowy, względem którego określane jest przejście (`SEEK_SET` - początek pliku, `SEEK_CUR` - bieżąca pozycja, `SEEK_END` - koniec pliku)
- Gdy wywołanie jest poprawne, to funkcja zwraca wartość `0` gdy wystąpił błąd (np. próba przekroczenia granic pliku), to funkcja zwraca wartość `-1`

## Przykład: odczytanie liczby o podanym numerze

```
#include <stdio.h>

int main(void)
{
    FILE *fp;
    int x, nr;

    fp = fopen("dane.dat", "rb");
    printf("Nr: "); scanf("%d", &nr);
    while (fseek(fp, (nr-1)*sizeof(int), SEEK_SET) == 0)
    {
        fread(&x, sizeof(int), 1, fp);
        printf("Liczba: %d\n", x);
        printf("Nr: "); scanf("%d", &nr);
    }
    printf("Koniec!\n");
    fclose(fp);
    return 0;
}
```

```
7 3 3 0 3 9 6 4 1 8
6 0 4 5 4 9 4 5 4 5
9 9 8 0 0 5 3 5 1 0
```

```
Nr: 6
Liczba: 9
Nr: 14
Liczba: 5
Nr: 29
Liczba: 1
Nr: -1
Koniec!
```

## Rekordowe (blokowe) operacje wejścia-wyjścia

```
FGETPOS stdio.h
int fgetpos(FILE *fp, fpos_t *pos);
```

- Zapamiętuje pod zmienną **pos** bieżące położenie w pliku wskazywanym przez **fp**; zwraca **0**, gdy wywołania jest poprawne i wartość niezerową, gdy wystąpił błąd

```
FSETPOS stdio.h
int fsetpos(FILE *fp, const fpos_t *pos);
```

- Przechodzi do położenia **pos** w pliku wskazywanym przez **fp**; zwraca **0**, gdy wywołania jest poprawne i wartość niezerową, gdy wystąpił błąd

Koniec wykładu nr 4

Dziękuję za uwagę!

(Następny wykład: 20.11.2017)