



Fundusze
Europejskie
Wiedza Edukacja Rozwój



Rzeczpospolita
Polska

Unia Europejska
Europejski Fundusz Społeczny



Wydział Elektryczny
Katedra Elektrotechniki Teoretycznej i Metrologii

Materiały do wykładu z przedmiotu:

Informatyka

Kod: EDS1A1 007

WYKŁAD NR 5

Opracował: dr inż. Jarosław Forenc

Białystok 2018

Materiały zostały opracowane w ramach projektu „PB2020 - Zintegrowany Program Rozwoju Politechniki Białostockiej” realizowanego w ramach Działania 3.5 Programu Operacyjnego Wiedza, Edukacja, Rozwój 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Społecznego.

Plan wykładu nr 5

- Przekazywanie argumentów do funkcji przez wartość i wskaźnik
- Operacje wejścia-wyjścia w języku C, strumienie
- Operacje na plikach
 - otwarcie pliku
 - zamknięcie pliku
- Typy operacji wejścia-wyjścia
 - znakowe
 - łańcuchowe
 - sformatowane
 - rekordowe (blokowe)

Przekazywanie argumentów do funkcji

```
#include <stdio.h>

void fun(int *wsk, int wart)
{
    *wsk = 20;
    wart = 20;
}

int main(void)
{
    int wsk = 10, wart = 10;

    printf("wsk: %d, wart: %d\n", wsk, wart);
    fun(&wsk, wart);
    printf("wsk: %d, wart: %d\n", wsk, wart);

    return 0;
}
```

```
wsk: 10, wart: 10
wsk: 20, wart: 10
```

Parametry funkcji - wektory

- Wektory przekazywane są do funkcji przez **wskaźnik**
- Nie jest tworzona kopia tablicy, a wszystkie operacje na jej elementach odnoszą się do tablicy z funkcji wywołującej
- W nagłówku funkcji podaje się typ elementów tablicy, jej nazwę oraz nawiasy kwadratowe z liczbą elementów tablicy lub same nawiasy kwadratowe

```
void fun(int tab[5])  
{  
    ...  
}
```

```
void fun(int tab[])  
{  
    ...  
}
```

- W wywołaniu funkcji podaje się tylko jej nazwę (bez nawiasów kwadratowych)

```
fun(tab);
```

Parametry funkcji - wektory (przykład)

```
#include <stdio.h>

void drukuj(int tab[])
{
    for (int i=0; i<5; i++)
        printf("%3d", tab[i]);
    printf("\n");
}

void zeruj(int tab[5])
{
    for (int i=0; i<5; i++)
        tab[i] = 0;
}
```

```
float srednia(int tab[])
{
    float sr = 0;
    int suma = 0;

    for (int i=0; i<5; i++)
        suma = suma + tab[i];

    sr = (float)suma / 5;

    return sr;
}
```

Parametry funkcji - wektory (przykład)

```
int main(void)
{
    int tab[5] = {1,2,3,4,5};
    float sred;

    drukuj(tab);

    sred = srednia(tab);
    printf("Srednia elementow: %g\n", sred);
    printf("Srednia elementow: %g\n", srednia(tab));

    zeruj(tab);
    drukuj(tab);

    return 0;
}
```

```
1 2 3 4 5
srednia elementow: 3
srednia elementow: 3
0 0 0 0 0
```

Parametry funkcji - macierze

- Macierze przekazywane są do funkcji przez **wskaźnik**
- W nagłówku funkcji podaje się typ elementów tablicy, jej nazwę oraz w nawiasach kwadratowych liczbę wierszy i kolumn lub tylko liczbę kolumn

```
void fun(int tab[2][3])  
{  
    ...  
}
```

```
void fun(int tab[][3])  
{  
    ...  
}
```

- W wywołaniu funkcji podaje się tylko jej nazwę (bez nawiasów kwadratowych)

```
fun(tab);
```

Parametry funkcji - macierze (przykład)

```
#include <stdio.h>

void zero(int tab[][3])
{
    for (int i=0; i<2; i++)
        for (int j=0; j<3; j++)
            tab[i][j] = 0;
}

void drukuj(int tab[2][3])
{
    for (int i=0; i<2; i++)
    {
        for (int j=0; j<3; j++)
            printf("%3d", tab[i][j]);
        printf("\n");
    }
}
```

```
int main(void)
{
    int tab[2][3] =
        {1, 2, 3, 4, 5, 6};

    drukuj(tab);
    zero(tab);
    printf("\n");
    drukuj(tab);

    return 0;
}
```


Parametry funkcji - macierze (przykład)

```
#include <stdio.h>

void zero(int tab[][3])
{
    for (int i=0; i<2; i++)
        for (int j=0; j<3; j++)
            tab[i][j] = 0;
}

void drukuj(int tab[2][3])
{
    for (int i=0; i<2; i++)
    {
        for (int j=0; j<3; j++)
            printf("%3d", tab[i][j]);
        printf("\n");
    }
}
```

```
int main
{
    int t
    {
        0 0 0
        0 0 0

        druku
        zero(
        printf("\n");
        drukuj(tab);

        return 0;
    }
}
```

1	2	3
4	5	6
0	0	0
0	0	0

Parametry funkcji - struktury

- Struktury przekazywane są do funkcji przez **wartość** (nawet jeśli daną składową jest tablica)

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

struct pkt
{
    float x, y;
};

float odl(struct pkt pkt1, struct pkt pkt2)
{
    return sqrt(pow(pkt2.x-pkt1.x, 2) +
                pow(pkt2.y-pkt1.y, 2));
}
```

Parametry funkcji - struktury (przykład)

```
int main(void)
{
    struct pkt p1 = {2,3};
    struct pkt p2 = {-2,1};
    float wynik;

    wynik = odl(p1,p2);

    printf("Punkt nr 1: (%g,%g)\n",p1.x,p1.y);
    printf("Punkt nr 2: (%g,%g)\n",p2.x,p2.y);
    printf("Odleglosc = %g\n",wynik);

    return 0;
}
```

```
Punkt nr 1: (2,3)
Punkt nr 2: (-2,1)
Odleglosc = 4.47214
```

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

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

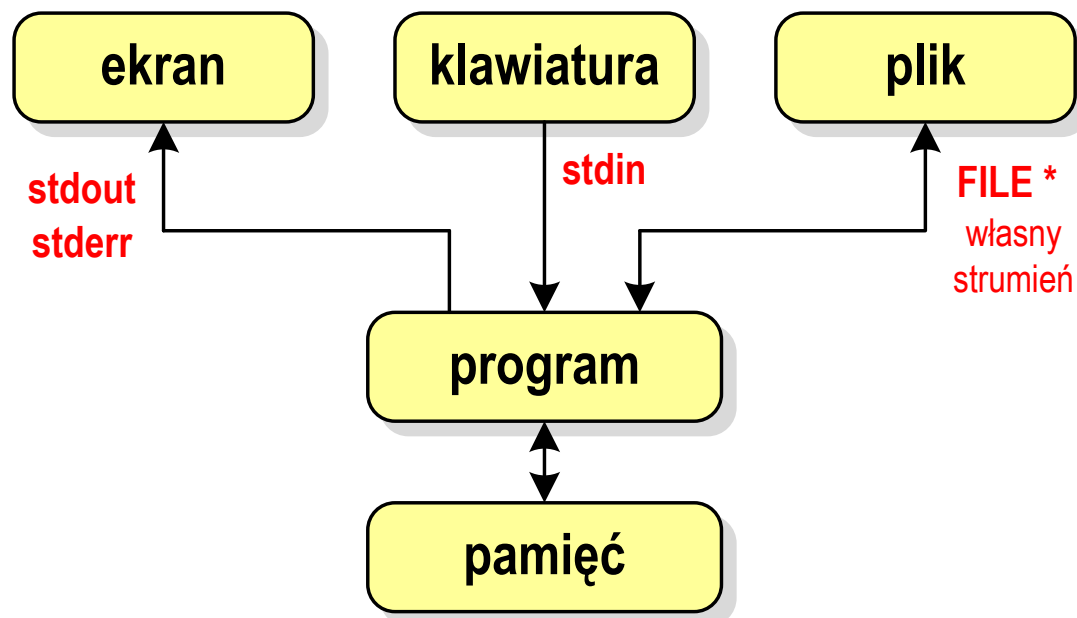
Strumienie

- 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])
```

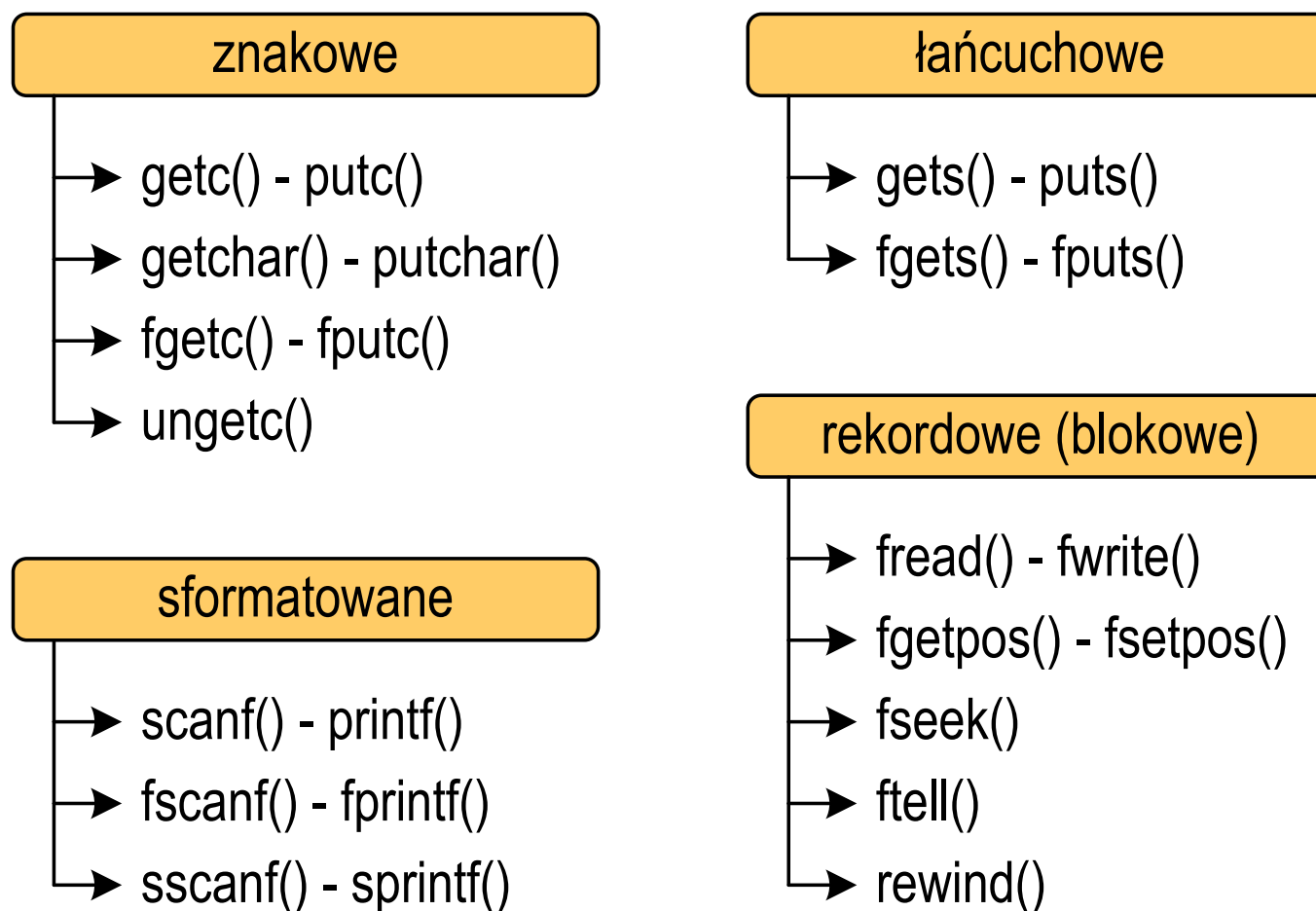
- Funkcja **printf()** niejawnie używa strumienia **stdout**
- Funkcja **scanf()** niejawnie używa strumienia **stdin**

Współpraca programu z „otoczeniem”

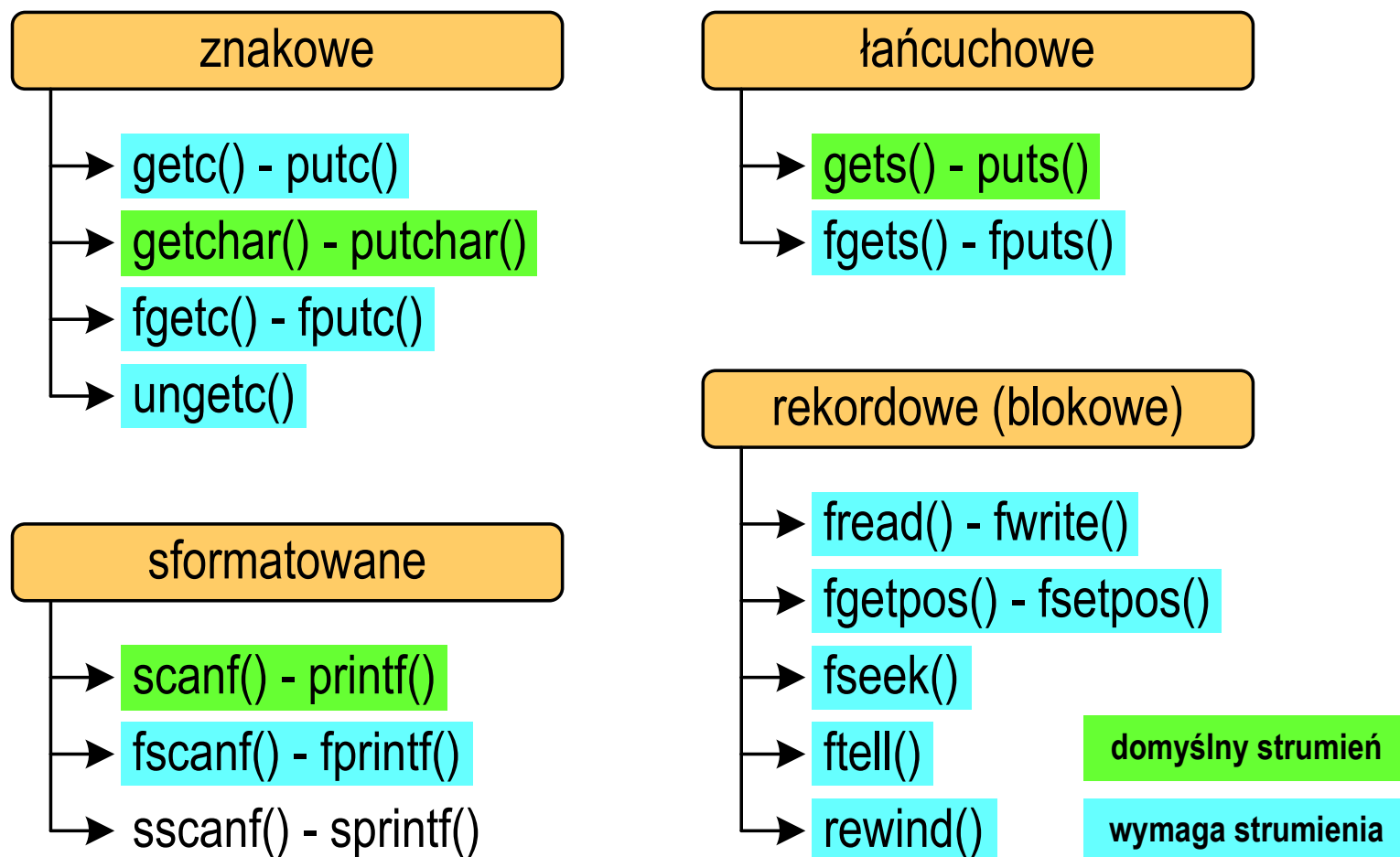


- Standardowe funkcje wejścia-wyjścia mogą:
 - domyślnie korzystać z określonego strumienia (**stdin**, **stdout**, **stderr**)
 - wymagać podania strumienia (własnego, **stdin**, **stdout**, **stderr**)

Typy standardowych operacji wejścia-wyjścia



Typy standardowych operacji wejścia-wyjścia



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");
```

Zamknięcie pliku - fclose()

FCLOSE

stdio.h

```
int fclose(FILE *fp);
```

- Zamyka plik wskazywany przez **fp**
- Zwraca **0 (zero)** jeśli zamknięcie pliku było pomyślne
- W przypadku wystąpienia błędu zwraca **EOF**

```
#define EOF      (-1)
```

- Po zamknięciu pliku, wskaźnik **fp** może być wykorzystany do otwarcia innego pliku
- W programie może być jednocześnie otwartych wiele plików

Przykład: otwarcie i zamknięcie pliku

```
#include <stdio.h>

int main(void)
{
    FILE *fp;

    fp = fopen("plik.txt", "w");
    if (fp == NULL)
    {
        printf("Bład otwarcia pliku.\n");
        return (-1);
    }

    /* przetwarzanie pliku */

    fclose(fp);

    return 0;
}
```


Format (plik) tekstowy i binarny

- Przykład zawartości pliku tekstowego (**Notatnik**):

Plik (ang. file) – uporządkowany zbiór danych o skończonej długości, posiadający szereg atrybutów i stanowiący dla użytkownika systemu operacyjnego całość. Nazwa pliku nie jest częścią tego pliku, lecz jest przechowywana w systemie plików.

- Przykład zawartości pliku binarnego (**Notatnik**):

```
MZ. . . . . @                Ć  .ššš ' .í! . .
LÍ!This program cannot be run in DOS mode....$  {9Āó?XF! ?XF! ?XF! ! .ō! <X
f! ! .í! ,XF! ↑ž. ! =XF! ?Xg! ˘XF! ! .â! 7XF! ! .ň! >XF! ! .÷! >XF! Rich?XF!
PE  L. • . ^ZR                ř  7 . σ . . 8 :  ↑◀ . + + @ + 7 |
|                               . @ . + + + + +   € . < .. $ |
. t.                            . .  W. . . . .                ř . text
```

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 **tekstowego** znajdzie się w nim 6 bajtów zawierających kody ASCII kolejnych cyfr

00110001	00110010	00110011	00110100	00110101	00110110
----------	----------	----------	----------	----------	----------

 (2)

'1'	'2'	'3'	'4'	'5'	'6'
-----	-----	-----	-----	-----	-----

 znaki

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 - $13_{(10)} = 0D_{(16)} = '\r'$
 - **LF** (line feed) - przesunięcie o wiersz, kod ASCII - $10_{(10)} = 0A_{(16)} = '\n'$
- 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 | pliku■■Drugi wie
72 73 7A 20 70 6C 69 6B|75 0D 0A 54 72 7A 65 63 | rsz pliku■■Trzec
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 - $10_{(10)} = 0A_{(16)} = '\n'$
- 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

```
FILE *fp1, *fp2;  
fp1 = fopen("dane.txt", "r"); // lub "rt"  
fp2 = fopen("dane.dat", "rb")
```

- 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

- `getc() - putc()`
- `getchar() - putchar()`
- `fgetc() - fputc()`
- `ungetc()`

sformatowane

- `scanf() - printf()`
- `fscanf() - fprintf()`
- `sscanf() - sprintf()`

łańcuchowe

- `gets() - puts()`
- `fgets() - fputs()`

rekordowe (blokowe)

- `fread() - fwrite()`
- `fgetpos() - fsetpos()`
- `fseek()`
- `ftell()`
- `rewind()`

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 `kodu` 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>

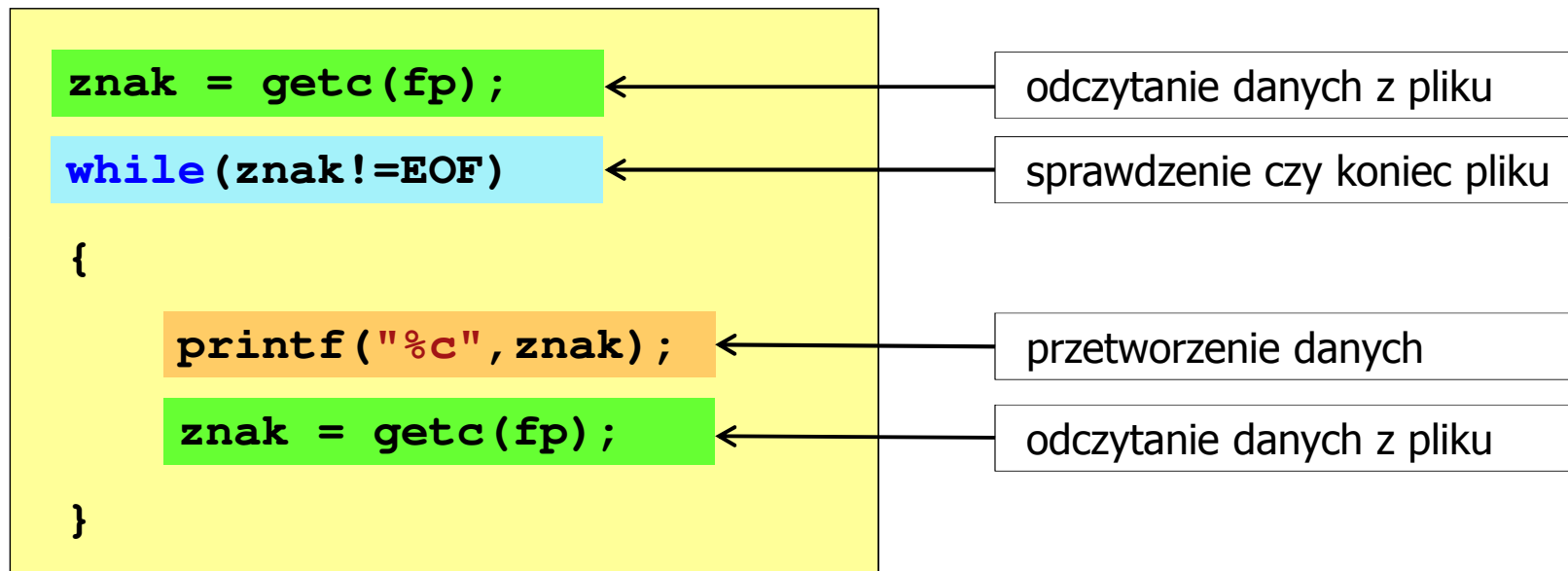
int main(void)
{
    FILE *fp;
    int znak;

    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



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**

```
for (int i='a'; i<='z'; i++)  
    putchar(i);
```

abcdefghijklmnopqrstuvwxyz

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;

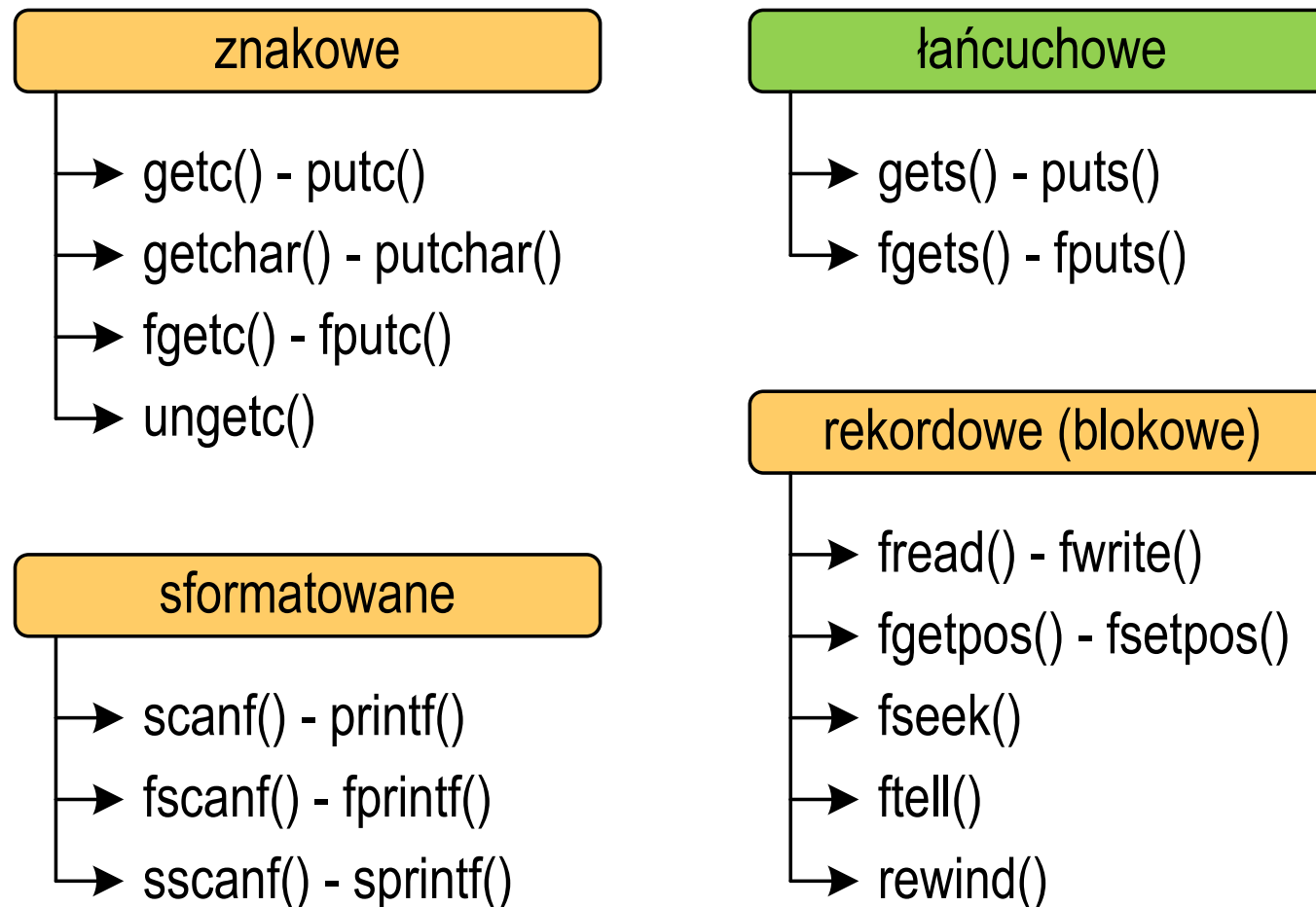
    fp = fopen("test.txt", "r");
    while ((znak = fgetc(fp)) != EOF)
        if (znak == ' ' || znak == '\t' || znak == '\n')
            odstep = 1;
        else
            if (odstep != 0) { odstep = 0; ile++; }
    fclose(fp);
    printf("Liczba slow: %d\n", ile);

    return 0;
}
```

Ala ma laptopa i psa.

Liczba slow: 5

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 C CR LF  
był język B, CR LF  
który CR LF  
Ritchie rozwinał w język C. CR LF
```

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

```
Poprzednikiem języka C CR LF  
był język B, CR LF  
który CR LF  
Ritchie rozwinał w język C. CR LF
```

Przykład: wyświetlenie pliku tekstowego

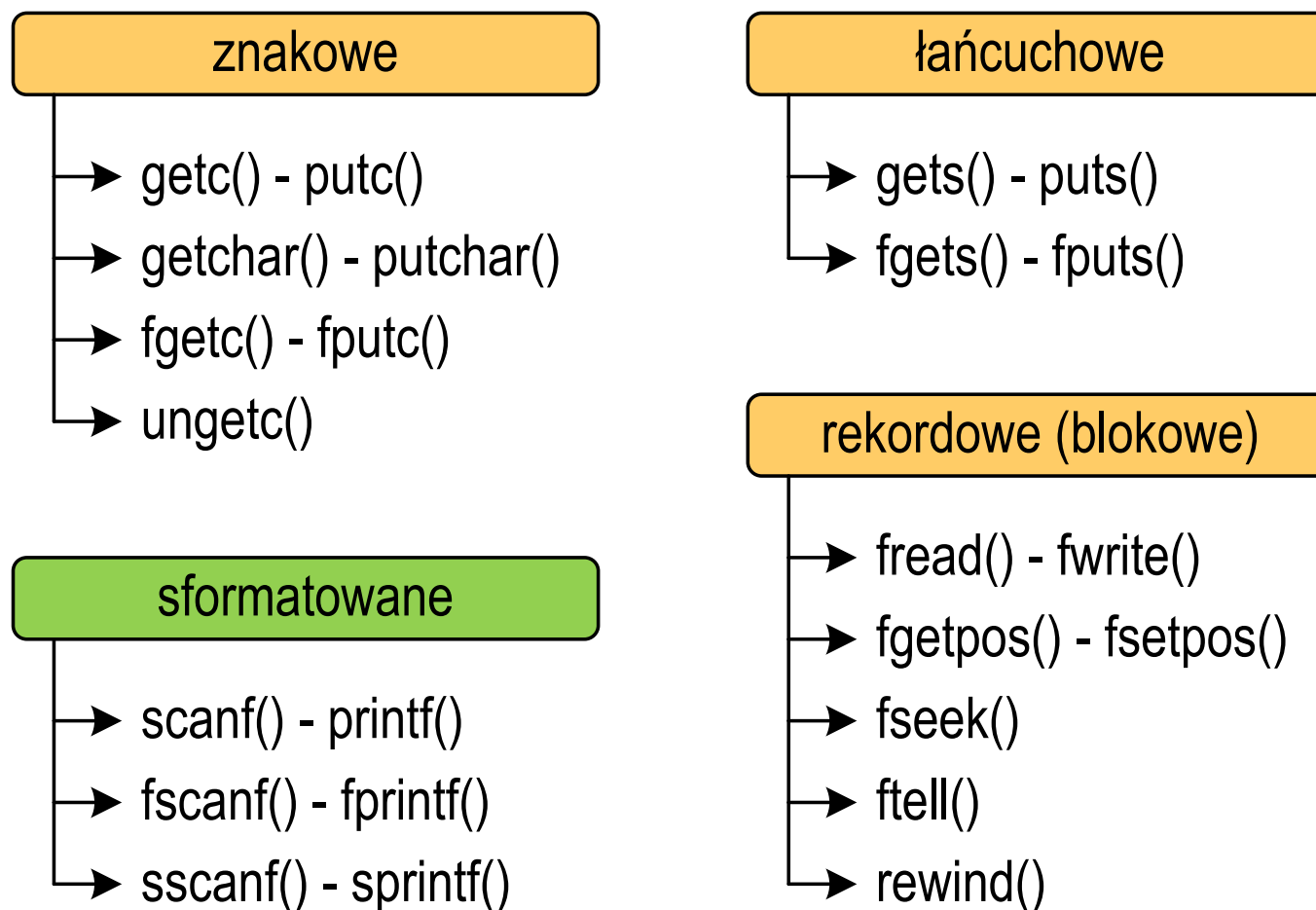
- Kolejne wywołania funkcji `fgets(buf,15,fp);` i zawartość tablicy `buf`

```
Poprzednikiem języka C CR LF  
był język B, CR LF  
który CR LF  
Ritchie rozwinał w język C. CR LF
```

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

`LF` = `\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 liczb do pliku tekstowego

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

int main(void)
{
    FILE *fp; float x; int i;

    srand((unsigned int)time(NULL));
    fp = fopen("liczby.txt", "w");
    for (i=0; i<10; i++)
    {
        x = (float)rand()/RAND_MAX*100;
        fprintf(fp, "%f\n", x);
    }
    fclose(fp);

    return 0;
}
```

```
3.830073
70.848717
99.322487
19.812616
7.132175
49.134800
10.238960
18.668173
8.914456
69.258705
```

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; float x;

    fp = fopen("liczby.txt", "r");

    fscanf(fp, "%f", &x);
    while (!feof(fp))
    {
        printf("%f\n", x);
        fscanf(fp, "%f", &x);
    }

    fclose(fp);

    return 0;
}
```

```
3.830073
70.848717
99.322487
19.812616
7.132175
49.134800
10.238960
18.668173
8.914456
69.258705
```

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

```
3.830073
70.848717
99.322487
19.812616
7.132175
49.134800
10.238960
18.668173
8.914456
69.258705
```

```
3.830073    70.848717
99.322487   19.812616
7.132175    49.134800
10.238960   18.668173
8.914456    69.258705
```

```
3.830073    70.848717    99.322487
19.812616   7.132175     49.134800
10.238960   18.668173    8.914456
69.258705
```

Przykład: odczytanie danych z pliku tekstowego

- Odczytanie danych różnych typów z pliku tekstowego

```
Nowak Grzegorz 15-12-2000  
Kowalski Wojciech 03-05-1997  
Jankowska Anna 23-05-1995  
Mazur Krzysztof 14-01-1990  
Krawczyk Monika 03-11-1995  
Piotrowska Maja 12-06-1998  
Dudek Piotr 31-12-1996  
Pawlak Julia 01-01-1997
```

Grzegorz	Nowak	wiek: 18
Wojciech	Kowalski	wiek: 21
Anna	Jankowska	wiek: 23
Krzysztof	Mazur	wiek: 28
Monika	Krawczyk	wiek: 23
Maja	Piotrowska	wiek: 20
Piotr	Dudek	wiek: 22
Julia	Pawlak	wiek: 21

Przykład: odczytanie danych z pliku tekstowego

```
#include <stdio.h>

int main()
{
    FILE *fp;
    char naz[20], im[20];
    int d, m, r;

    fp = fopen("osoby.txt", "r");
    fscanf(fp, "%s %s %d-%d-%d", naz, im, &d, &m, &r);
    while (!feof(fp))
    {
        printf("%-12s %-12s wiek: %d\n", im, naz, 2018-r);
        fscanf(fp, "%s %s %d-%d-%d", naz, im, &d, &m, &r);
    }
    fclose(fp);

    return 0;
}
```

Przykład: odczytanie danych z pliku tekstowego

```
#include <stdio.h>

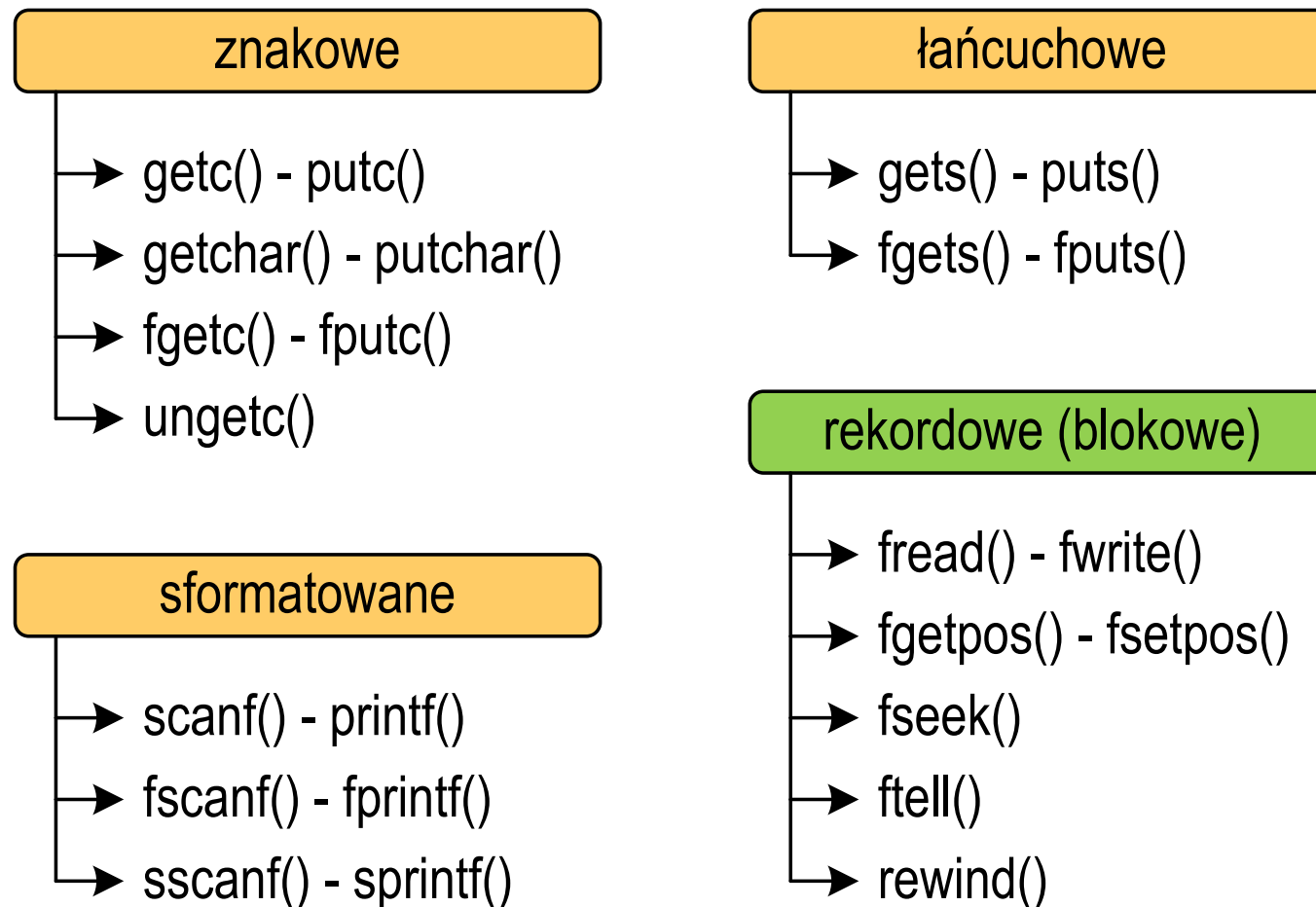
int main()
{
    FILE *fp;
    char naz[20], im[20];
    int d, m, r;

    fp = fopen("osoby.txt", "r");
    fscanf(fp, "%s %s %d-%d-%d", naz, im, &d, &m, &r);
    while (!feof(fp))
    {
        printf("%-12s %-12s wiek: %d\n", im, naz, 2018-r);
        fscanf(fp, "%s %s %d-%d-%d", naz, im, &d, &m, &r);
    }
    fclose(fp);

    return 0;
}
```

Grzegorz	Nowak	wiek: 18
Wojciech	Kowalski	wiek: 21
Anna	Jankowska	wiek: 23
Krzysztof	Mazur	wiek: 28
Monika	Krawczyk	wiek: 23
Maja	Piotrowska	wiek: 20
Piotr	Dudek	wiek: 22
Julia	Pawlak	wiek: 21

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);

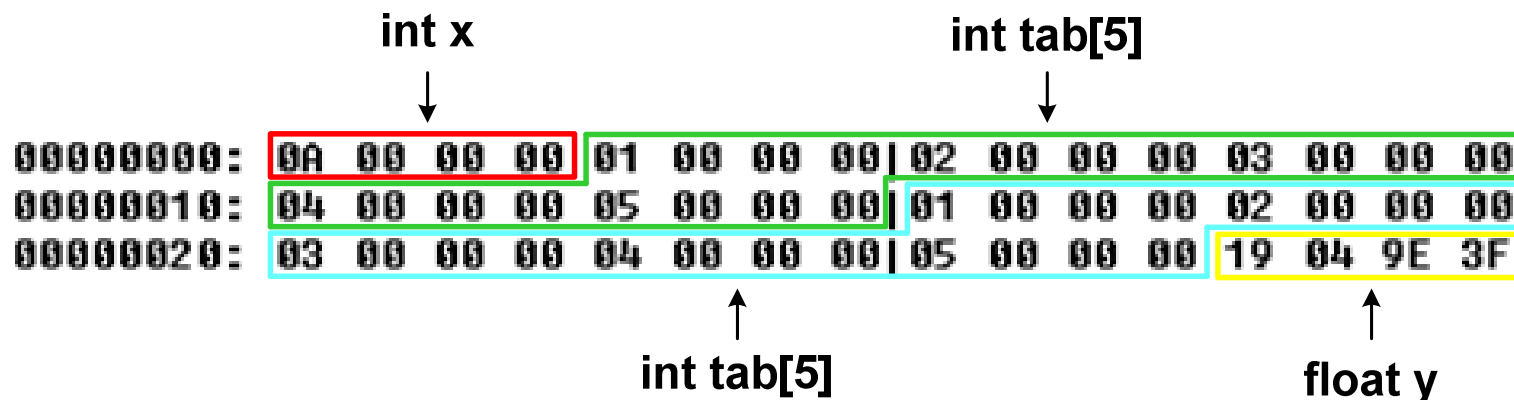
    return 0;
}
```


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("Wystąpienia 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 5

Dziękuję za uwagę!