

Informatyka 1 (ES1F1002)

Politechnika Białostocka - Wydział Elektryczny
Elektrotechnika, semestr I, studia stacjonarne I stopnia
Rok akademicki 2023/2024

Wykład nr 13 (09.01.2024)

dr inż. Jarosław Forenc

Plan wykładu nr 13

- Funkcje w języku C
 - przekazywanie argumentów przez wartość i przez wskaźnik
 - przekazywanie wektorów, macierzy i struktur do funkcji
- Pamięć a zmienne w programie
 - zmienne: automatyczne (auto), rejestrowe (register), zewnętrzne (extern), statyczne (static)
 - struktura procesu w pamięci komputera, ramka stosu
- Operacje wejścia-wyjścia w języku C, pliki
 - strumienie, typy standardowych operacji wejścia-wyjścia
 - operacje na plikach: otwarcie i zamknięcie pliku
 - format (plik) / tryb otwarcia tekstowy i binarny
 - operacje znakowe, łańcuchowe, sformatowane, rekordowe (blokowe)

Przekazywanie argumentów do funkcji

- Przekazywanie argumentów przez **wartość**:
 - po wywołaniu funkcji tworzone są lokalne kopie zmiennych skojarzonych z jej argumentami
 - w funkcji widoczne są one pod postacią parametrów funkcji
 - parametry te mogą być traktowane jak lokalne zmienne, którym przypisano początkową wartość

- Przekazywanie argumentów przez **wskaźnik**:
 - do funkcji przekazywane są adresy zmiennych będących jej argumentami
 - wszystkie operacje wykonywane w funkcji na takich argumentach będą odnosiły się do zmiennych z funkcji wywołującej

Przekazywanie argumentów do funkcji

```
#include <stdio.h>

void fff(int kopiax, int *wsky)
{
    kopiax = 20;
    *wsky = 20;
}

int main(void)
{
    int x = 10, y = 10;

    fff(x, &y);
    printf("x = %d\n", x);
    printf("y = %d\n", y);

    return 0;
}
```

```
x = 10
y = 20
```

- Przekazywanie argumentów do funkcji:
x - przez **wartość**
y - przez **wskaźnik**

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] = {9,3,6,5,1};
    float sred;

    drukuj(tab);

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

    zeruj(tab);
    drukuj(tab);

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

    return 0;
}
```

```
9 3 6 5 1
Srednia: 4.800000
0 0 0 0 0
Srednia: 0.000000
```

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
    {
        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
```

Pamięć a zmienne w programie

- Ze względu na czas życia wyróżnia się w programie:
 - **obiekty statyczne** - istnieją od chwili rozpoczęcia działania programu aż do jego zakończenia
 - **obiekty dynamiczne** - tworzone i usuwane z pamięci w trakcie wykonania programu
 - automatycznie (bez udziału programisty)
 - kontrolowane przez programistę
- O typie obiektu (**statyczny** lub **dynamiczny**) decyduje klasa pamięci obiektu (ang. storage class)
 - **auto** - zmienne automatyczne
 - **register** - zmienne umieszczane w rejestrach procesora
 - **extern** - zmienne zewnętrzne
 - **static** - zmienne statyczne

Zmienne automatyczne - auto

- Miejsce deklaracji: najczęściej początek bloku funkcyjnego ograniczonego nawiasami klamrowymi { i }
- Pamięć przydzielana automatycznie przy wejściu do bloku i zwalniana po wyjściu z niego
- Zakres widzialności: ograniczony do bloku, w którym zmienne zostały zadeklarowane (**zmienne lokalne**)
- Dostęp do zmiennych z innych bloków możliwy przez wskaźnik
- Jeśli zmienne są inicjalizowane, to odbywa się ona przy każdym wejściu do bloku, w którym zostały zadeklarowane
- Nie ma potrzeby jawnego używania **auto**, gdyż domyślnie zmienne wewnątrz bloków funkcyjnych są lokalne

```
auto int x;
```

Zmienne rejestrowe - register

- Zazwyczaj o miejscu umieszczenia zmiennej automatycznej decyduje kompilator:
 - pamięć operacyjna - wolniejszy dostęp
 - rejestry procesora - szybszy dostęp
- Programista może zasugerować kompilatorowi umieszczenie określonej zmiennej automatycznej w rejestrach procesora
- Najczęściej dotyczy to zmiennych:
 - często używanych
 - takich, dla których czas dostępu jest bardzo ważny

```
register int x;
```

Zmienne zewnętrzne - extern

- Miejsce deklaracji: poza blokami funkcyjnymi, najczęściej na początku pliku z kodem źródłowym
- Pamięć na zmienne jest przydzielana, gdy program rozpoczyna pracę i zwalniana, gdy program kończy się
- Zakres widzialności: globalny - od miejsca deklaracji do końca pliku z kodem źródłowym (**zmienne globalne**)
- Jeśli inna zmienna lokalna, ma taką samą nazwę jak globalna, to lokalna przesłania widoczność zmiennej globalnej
- W większości implementacji języka C zmienne **extern** są automatycznie inicjalizowane zerem
- Etykieta **extern** może być pominięta (chyba, że program składa się z kilku plików z kodem źródłowym)
- Zalecane jest ograniczenie stosowania zmiennych globalnych

Zmienne statyczne - static

- Miejsce deklaracji: w bloku funkcyjnym jako automatyczne lub poza blokami funkcyjnymi, jako globalne
- Istnieją przez cały czas wykonywania programu, nawet po zakończeniu bloku funkcyjnego, w którym zostały zadeklarowane
- Zakres widzialności: zależny od sposobu deklaracji (automatyczne lub globalne)
- Zmienne **static** są automatycznie inicjalizowane zerem
- Mogą być inicjalizowane podczas deklaracji (tylko stałą wartością), inicjalizacja jest wykonywana tylko raz, podczas kompilacji programu

```
static int x = 10;
```

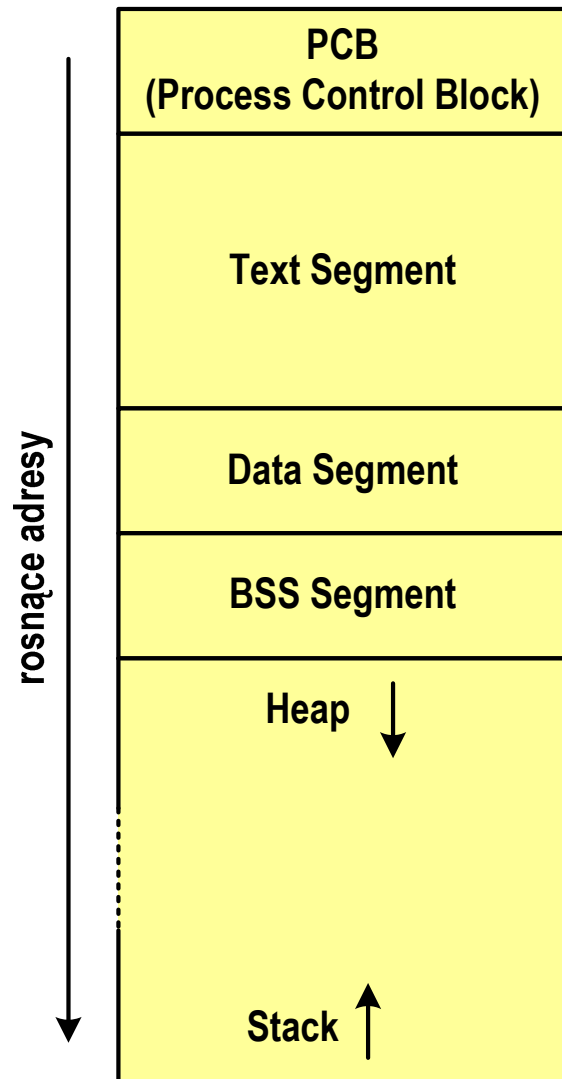
Klasy pamięci zmiennych

```
int a;                /* extern int a; - zmienna globalna */
void foo();

int main(void)
{
    int b;            /* auto int b; - zmienna lokalna */
    register float a; /* zmienna automatyczna, rejestrowa */
    foo(); foo(); foo();
    return 0;
}

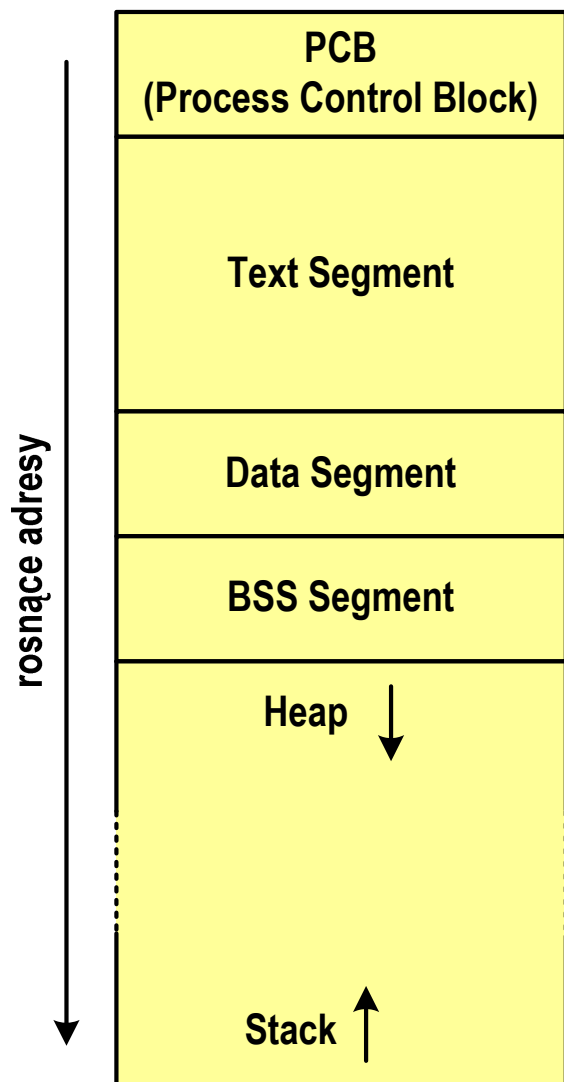
void foo()
{
    static int c = 1; /* zmienna statyczna */
    {
        double a;    /* zmienna lokalna */
    }
    c++; a++;
}
```

Struktura procesu w pamięci komputera



- **PCB** - blok kontrolny procesu
 - obszar pamięci operacyjnej zarezerwowany przez system operacyjny do zarządzania procesem
- **Text Segment**
 - kod programu czyli instrukcje w postaci binarnej
- **Data Segment**
 - zmienne globalne i statyczne zainicjalizowane niezerowymi wartościami
- **BSS Segment** (Block Started by Symbol)
 - zmienne globalne i statyczne domyślnie zainicjalizowane zerowymi wartościami

Struktura procesu w pamięci komputera



- **Heap** - sverta
 - obszar zmiennych dynamicznych
 - pamięć w obszarze sterty przydzielana jest funkcjami `calloc()` i `malloc()`
- **Stack** - stos
 - zmienne lokalne (automatyczne)
 - parametry funkcji i adresy powrotu z funkcji (stack frame)

Zmienne w pamięci komputera

```
int a;                                /* BSS Segment */
void foo();

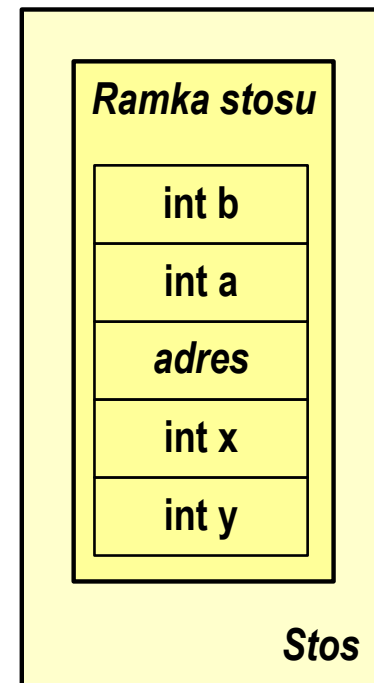
int main(void)
{
    int b;                             /* Stack */
    float *a;                          /* Stack */
    a = (float *) malloc(400);         /* Heap - 400 bajtów */
    return 0;
}

void foo()
{
    static int c = 1;                 /* Data Segment */
    {
        double a;                    /* Stack */
    }
    c++; a++;
}
```

Ramka stosu (stack frame)

- Każde wywołanie funkcji powoduje odłożenie na stosie tzw. **ramki stosu**

```
void fun(int x, int y)
{
    int a, b;
}
```



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
- Strumień może być skojarzony ze zbiorem danych znajdujących się na dysku (plik) lub zbiorem danych pochodzących z urządzenia znakowego (klawiatura)
- Strumienie reprezentowane są przez zmienne będące wskaźnikami na struktury typu **FILE** (definicja w pliku **stdio.h**)
- Podczas pisania programów nie ma potrzeby bezpośredniego odwoływania się do pól tej struktury

Strumienie

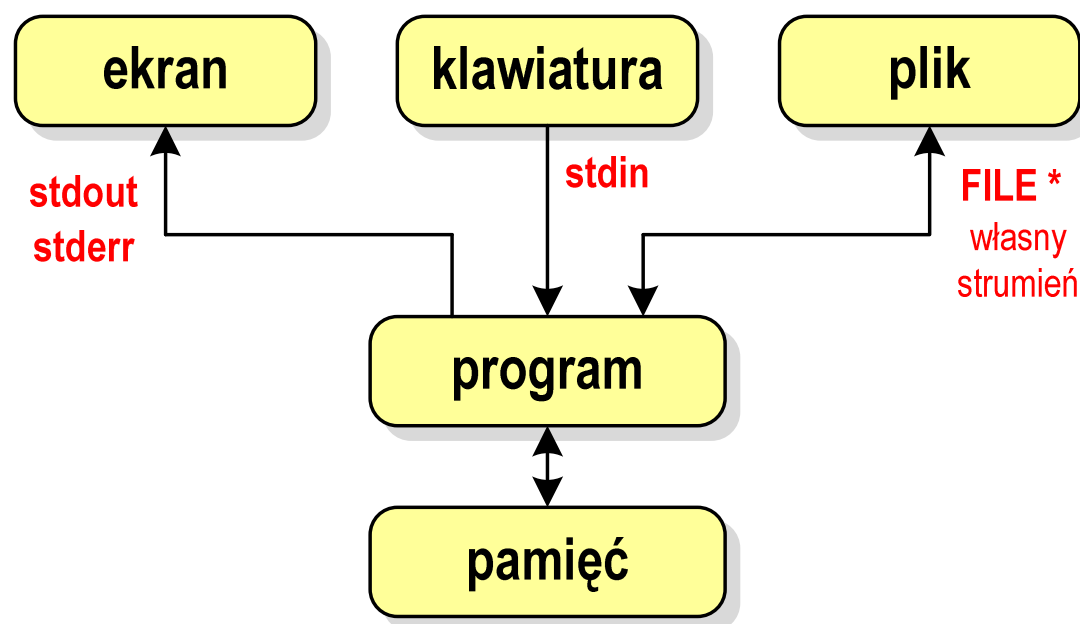
- W każdym programie automatycznie tworzone są i otwierane trzy standardowe strumienie wejścia-wyjścia:
 - **stdin** - standardowe wejście, skojarzone z klawiaturą
 - **stdout** - standardowe wyjście, skojarzone z ekranem monitora
 - **stderr** - standardowe wyjście dla komunikatów o błędach, skojarzone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

Strumienie

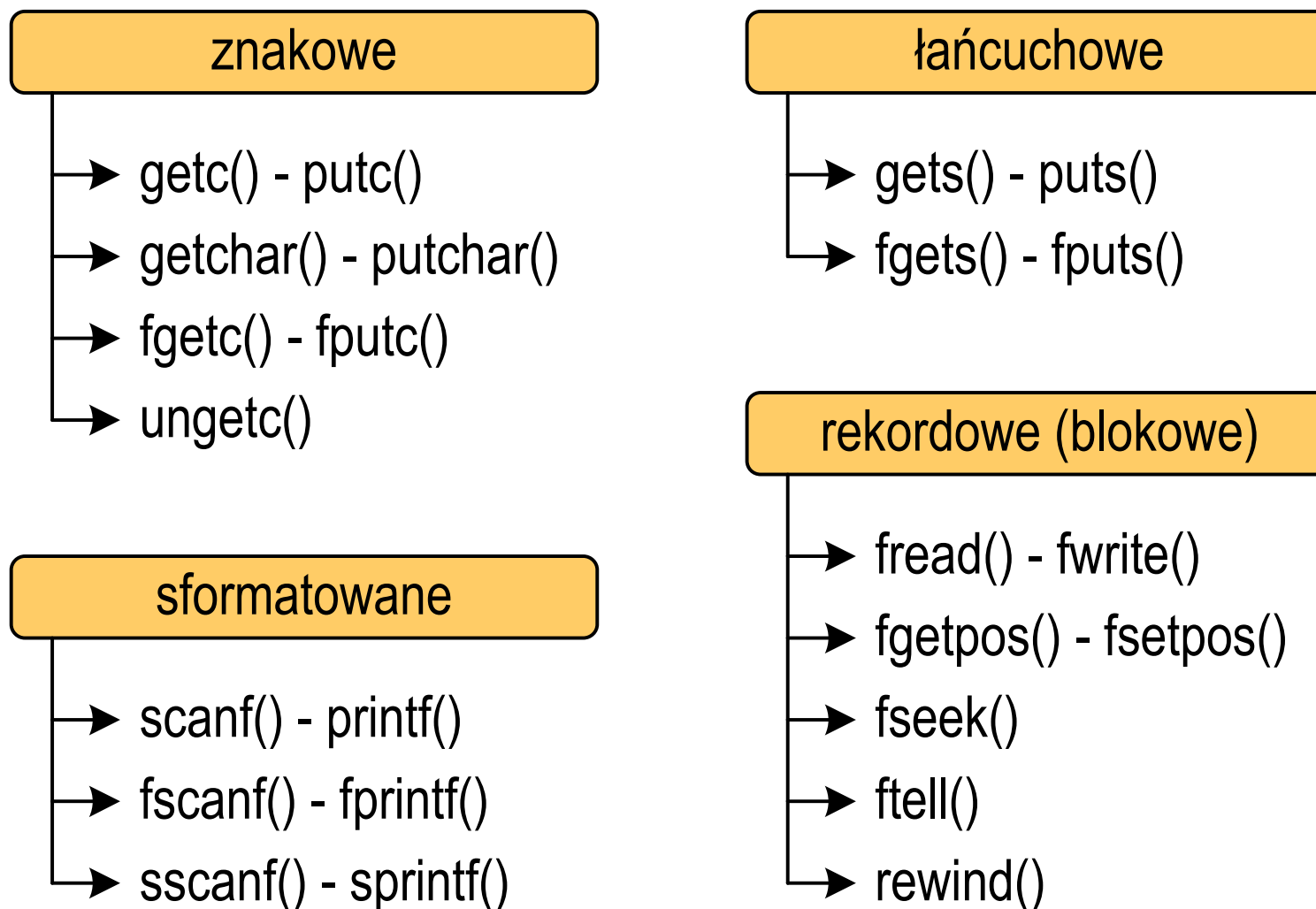
■ Współpraca programu z „otoczeniem”



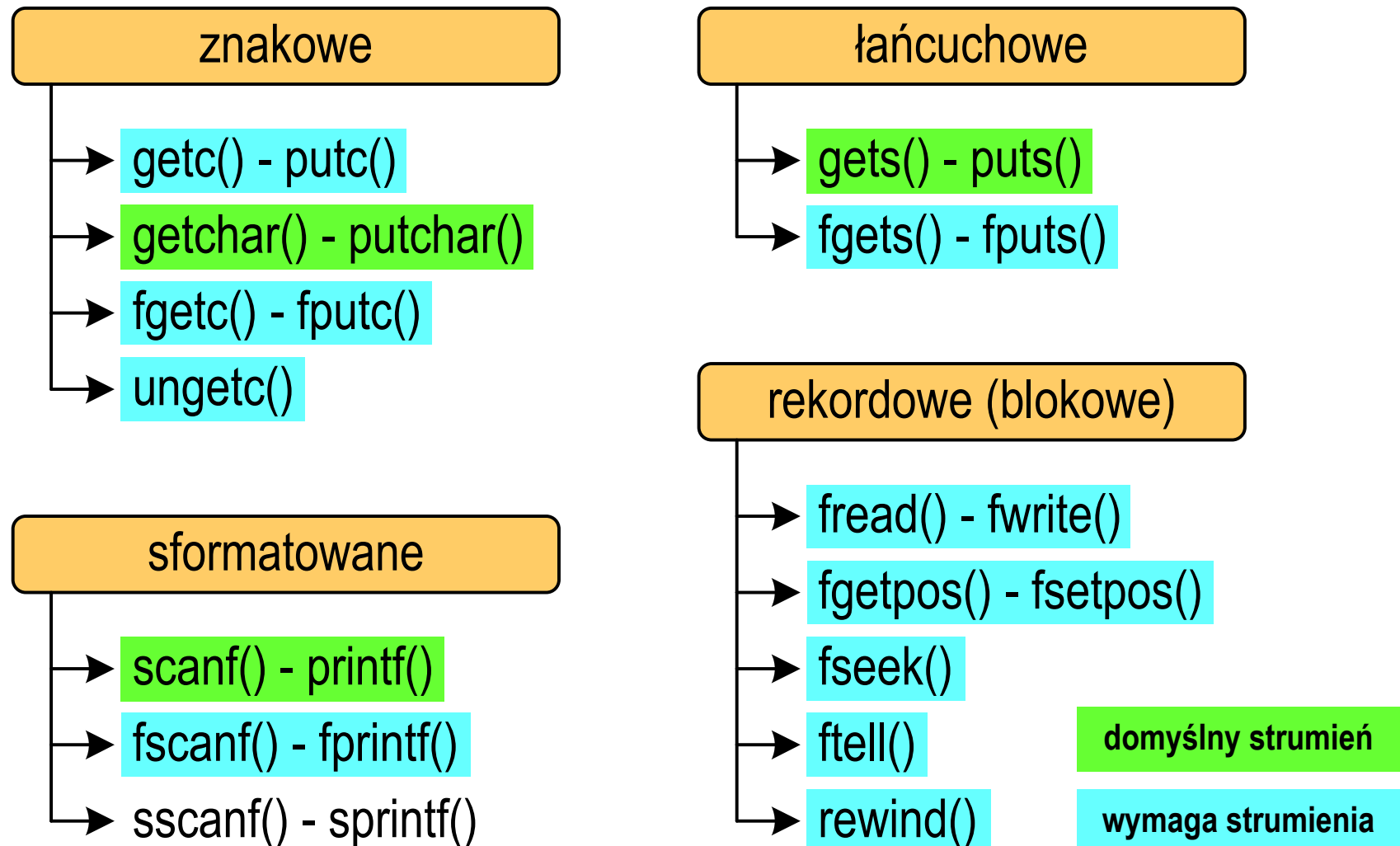
■ Standardowe funkcje wejścia-wyjścia mogą:

- 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()

```
FILE* fopen(const char *fname, const char *mode);
```

- **fname** - nazwa pliku, może zawierać całą ścieżkę dostępu do pliku
- **mode** - tryb otwarcia:
 - **"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
 - **"t"** - otwarcie w trybie tekstowym (domyślnie)
 - **"b"** - otwarcie w trybie binarnym
- **fopen()** zwraca wskaźnik na strukturę **FILE** skojarzoną z otwartym plikiem lub **NULL**, gdy otwarcie nie powiodło się

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. L J .. @ c . 5 8 ' . Í ! , .  
LÍ!This program cannot be run in DOS mode....$ {90ó?XF|?XF|?XF|! .ó|<X  
f|! .Í| ,XF|! ž .|=XF|?Xg| LXF|! .â|7XF|! .ň|>XF|! .÷|>XF|Rich?XF|  
PE L . • . ^ZR f 7 . σ .. 8 : ↑ ◀ . + + @ + 7 | $ |  
| ° . J L @ . + + + + + € . < .. $ |  
 . t . . □ L W . . .textbss . + r .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** znajdą 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 | pliku■Drugi wier
73 7A 20 70 6C 69 6B 75|0A 54 72 7A 65 63 69 20 | sz pliku■Trzeci
77 69 65 72 73 7A 20 70|6C 69 6B 75 0A | wiersz pliku■
```

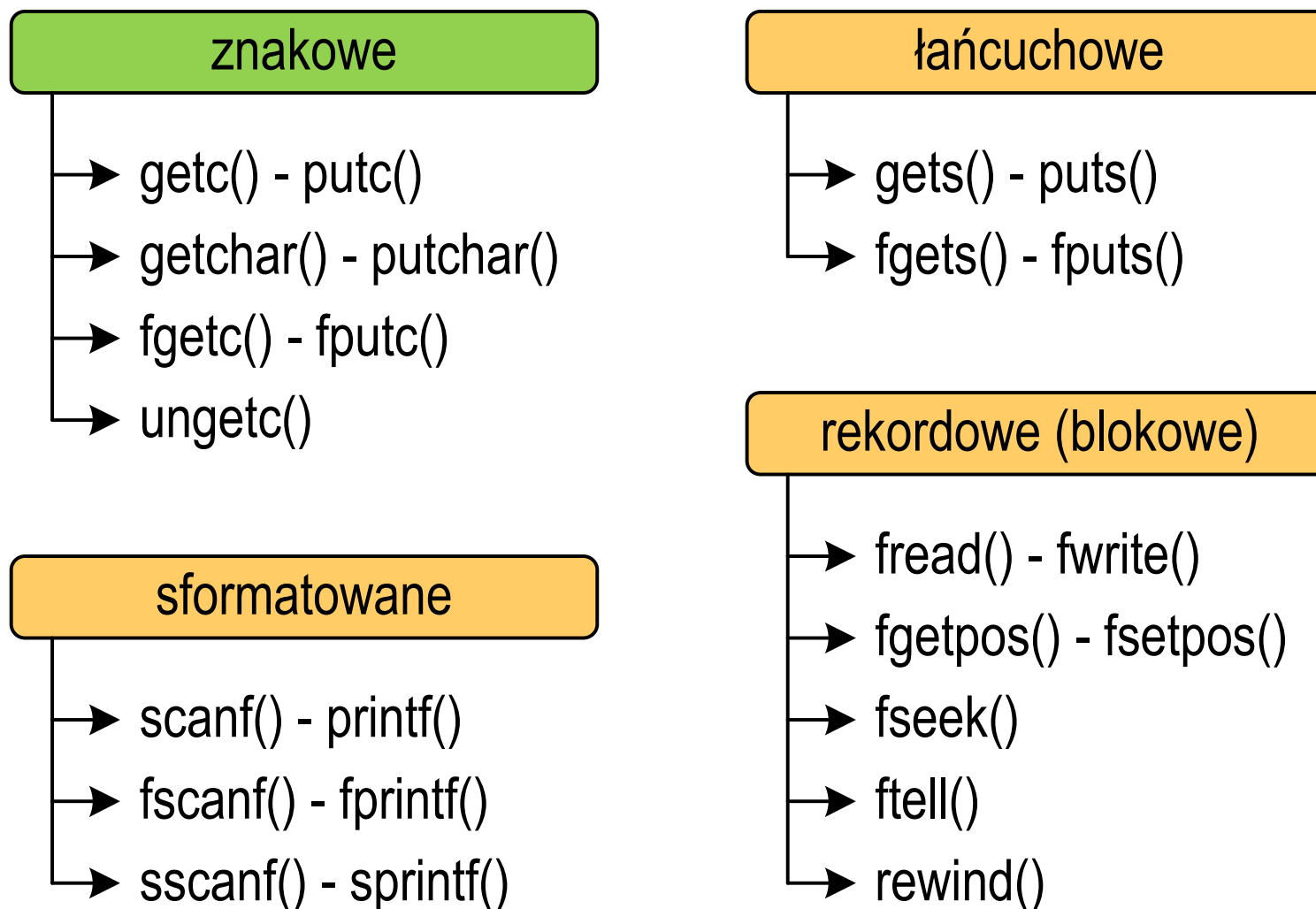
- Pliki **binarne** nie mają ściśle określonej struktury

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 operacje wejścia-wyjścia

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

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

- pobiera (czyta) jeden znak ze strumienia `fp` i zwraca jego kod (jako `int`) lub `EOF` (gdy napotkano koniec pliku)

```
int getchar();
```

- pobiera (czyta) jeden znak z klawiatury (strumienia `stdin`) i zwraca jego kod (jako `int`)

```
FILE *fp; int zn;  
zn = getc(fp);           // z pliku  
zn = fgetc(fp);         // z pliku  
zn = getchar();         // z klawiatury
```

Przykład: wyświetlenie pliku tekstowego

```
#include <stdio.h>

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

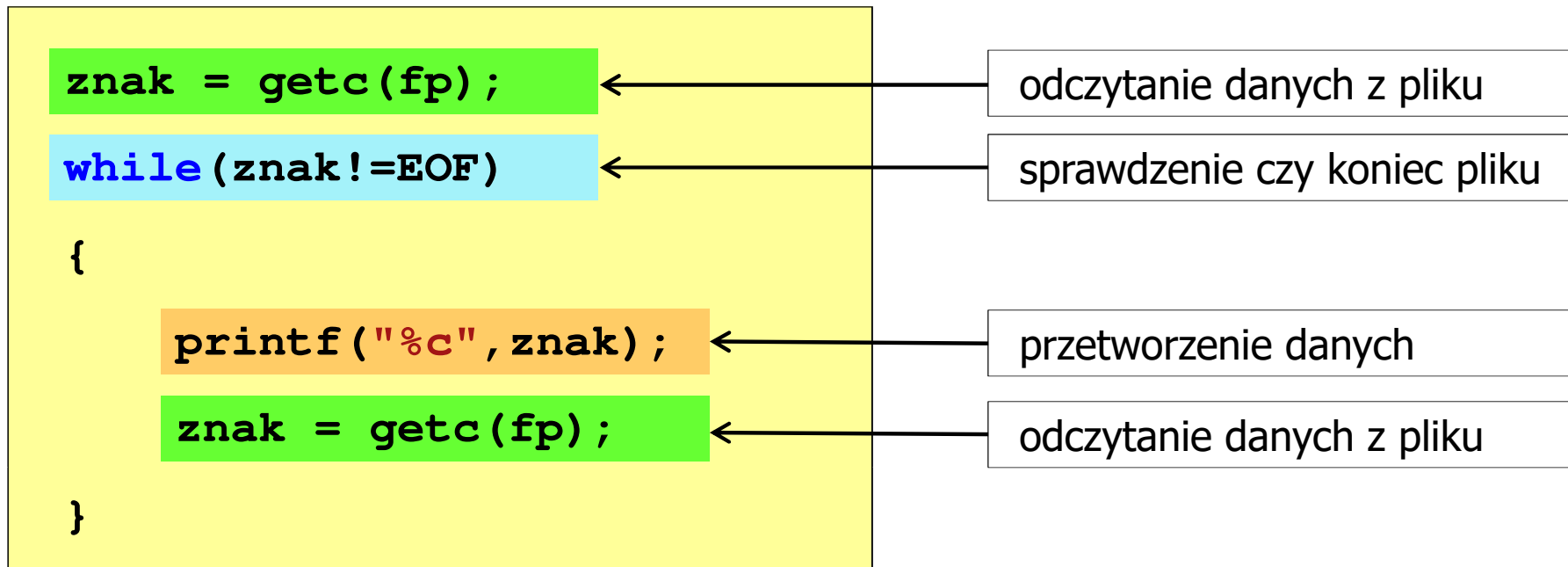
    fp = fopen("test.txt", "r");

    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

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

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

- wpisuje **znak** do otwartego strumienia **fp**

```
int putchar(int znak);
```

- wyświetla **znak** na ekranie (wpisuje do strumienia **stdout**)

```
FILE *fp; int zn = 'a';  
putc(zn, fp);           // do pliku  
fputc(zn, fp);         // do pliku  
putchar(zn);           // na ekran
```

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++)  
        putchar(i, fp);
```

```
    fclose(fp);
```

```
    return 0;
```

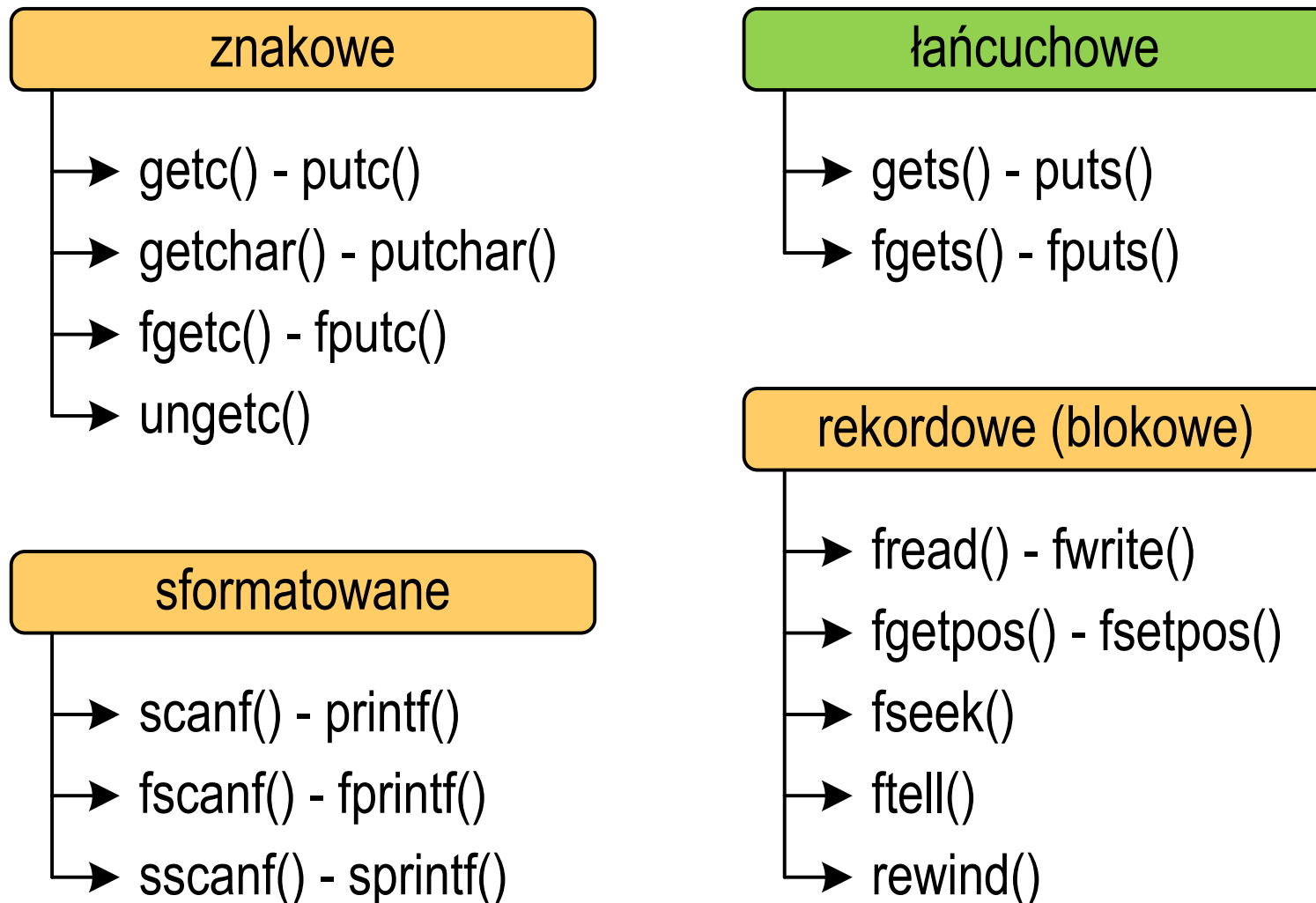
```
}
```

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- Stosując strumień `stdout` można wyświetlić alfabet na ekranie

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

Łańcuchowe operacje wejścia-wyjścia



Łańcuchowe operacje wejścia-wyjścia

```
char* gets(char *buf);
```

- czyta linię znaków z klawiatury (strumienia `stdin`) i zapisuje w tablicy `buf`; wczytywanie jest kończone po napotkaniu `\n`, który zastępowany jest znakiem `\0`

```
char* fgets(char *buf, int max, FILE *fp);
```

- czyta znaki z otwartego strumienia `fp` i zapisuje je w tablicy `buf`; przerywa pobieranie znaków po odczytaniu `\n` lub `max-1` znaków; zwraca `NULL` po napotkaniu końca pliku

```
FILE *fp; char txt[20];  
gets(txt);           // z klawiatury  
fgets(txt, 20, fp); // z pliku
```

Łańcuchowe operacje wejścia-wyjścia

```
int puts(const char *buf);
```

- wyświetla łańcuch znaków `buf` na ekranie (wpisuje do strumienia `stdout`);
zastępuje znak `\0` znakiem `\n`

```
int fputs(const char *buf, FILE *fp);
```

- wpisuje znaki z tablicy `buf` do otwartego strumienia `fp`;
nie dołącza znaku końca wiersza `\n`

```
FILE *fp; char txt[20] = "Witaj swiecie";  
puts(txt);           // na ekran  
fputs(txt, fp);     // do pliku
```


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 jezyka C CR LF  
byl jezyk B, CR LF  
ktory CR LF  
Ritchie rozwinal w jezyk C. CR LF
```

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

```
Poprzednikiem jezyka C CR LF  
byl jezyk B, CR LF  
ktory CR LF  
Ritchie rozwinal w jezyk C. CR LF
```

Przykład: wyświetlenie pliku tekstowego

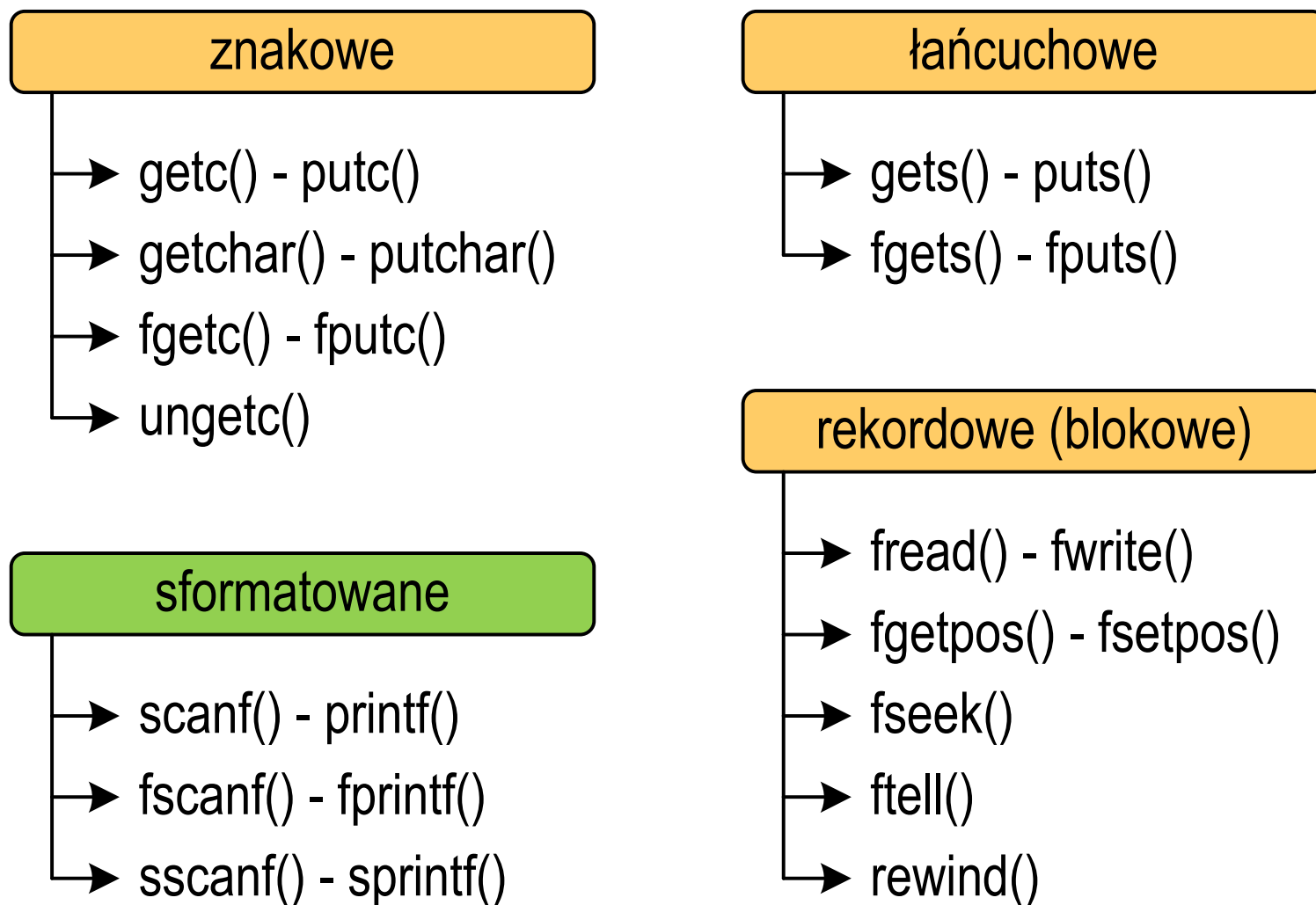
- Kolejne wywołania funkcji `fgets(buf,15,fp);` i zawartość tablicy `buf`

```
Poprzednikiem języka C  
był język B,  
ktory  
Ritchie rozwinał w język C.
```

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

LF = \n

Sformatowane operacje wejścia-wyjścia



Sformatowane operacje wejścia-wyjścia

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

- wyświetla dane na ekranie (wyprowadza do strumienia `stdout`)

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

- wyprowadza dane do otwartego strumienia `fp` (najczęściej pliku)

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

- zapisuje dane do tablicy znaków `buf`

```
FILE *fp; char txt[30];  
printf("Witaj swiecie"); // na ekran  
fprintf(fp, "Witaj swiecie"); // do pliku  
sprintf(txt, "Witaj swiecie"); // do tablicy znaków
```

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]
```

Sformatowane operacje wejścia-wyjścia

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

- czyta dane z klawiatury (strumienia `stdin`)

```
int fscanf(FILE *fp, const char *format, ...);
```

- czyta dane z otwartego strumienia `fp` (najczęściej pliku)

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

- czyta dane z tablicy znaków `buf`

```
FILE *fp; char txt[30] = "15 3.14"; int x; float y;  
scanf("%d %f", &x, &y); // z klawiatury  
fscanf(fp, "%d %f", &x, &y); // z pliku  
sscanf(txt, "%d %f", &x, &y); // z tablicy znaków
```

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: 24
Wojciech     Kowalski  wiek: 27
Anna         Jankowska wiek: 29
Krzysztof    Mazur     wiek: 34
Monika       Krawczyk  wiek: 29
Maja        Piotrowska wiek: 26
Piotr        Dudek     wiek: 28
Julia       Pawlak    wiek: 27
```


Wykrycie końca pliku

Funkcje	Metoda
getc(), fgetc()	zwracana wartość: EOF
fgets()	zwracana wartość: NULL
fscanf()	wywołanie funkcji: feof()

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

- zwraca wartość różną od zera, jeśli podczas ostatniej operacji odczytu pliku wskazywanego przez **fp** został wykryty jego koniec; w przeciwnym razie zwraca wartość **0** (zero)

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, 2024-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, 2024-r);
```

```
        fscanf(fp, "%s %s %d-%d-%d", naz, im, &d, &m, &r);
```

```
    }
```

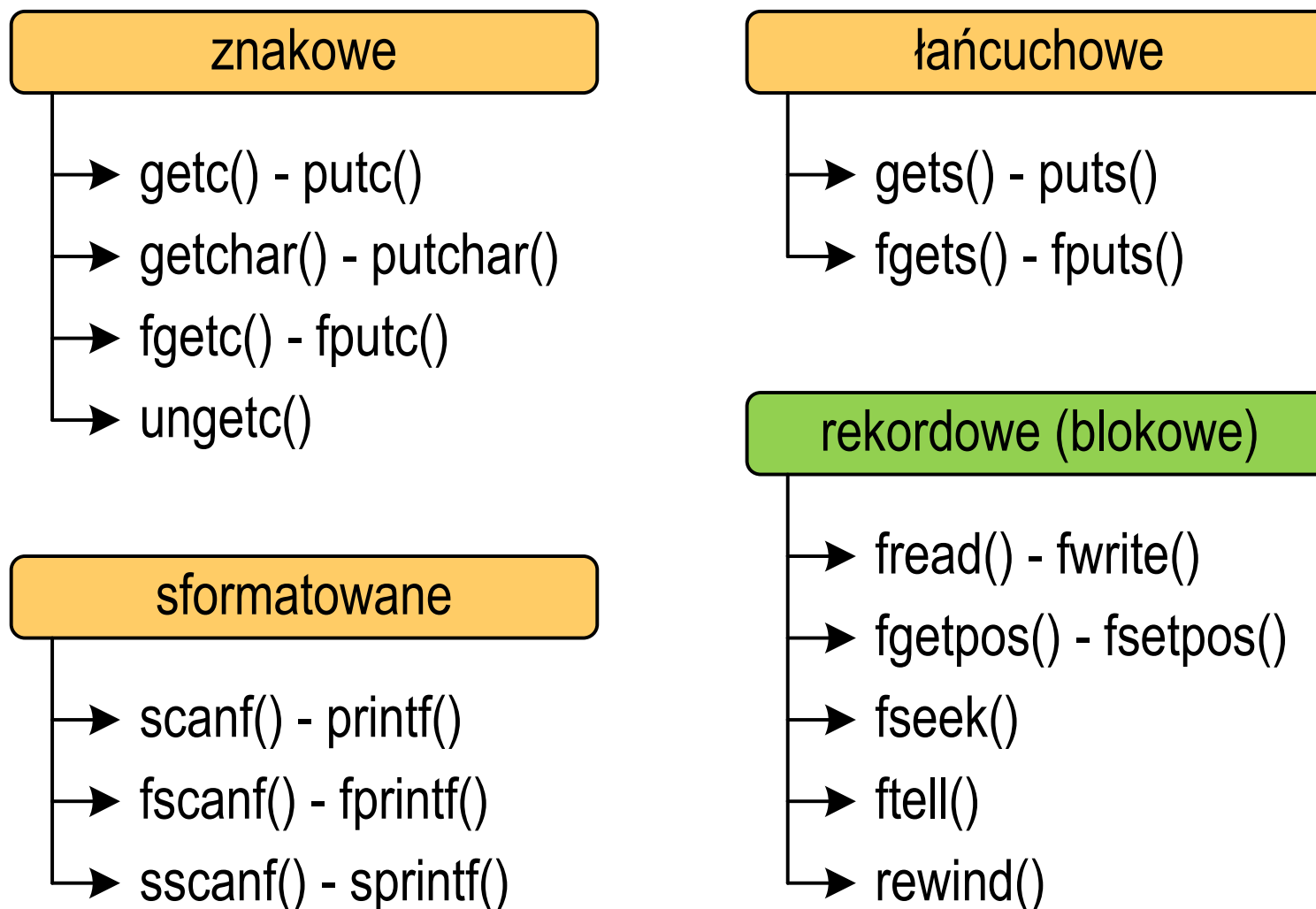
```
    fclose(fp);
```

```
    return 0;
```

```
}
```

Grzegorz	Nowak	wiek: 24
Wojciech	Kowalski	wiek: 27
Anna	Jankowska	wiek: 29
Krzysztof	Mazur	wiek: 34
Monika	Krawczyk	wiek: 29
Maja	Piotrowska	wiek: 26
Piotr	Dudek	wiek: 28
Julia	Pawlak	wiek: 27

Rekordowe (blokowe) operacje wejścia-wyjścia



Rekordowe (blokowe) operacje wejścia-wyjścia

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

- zapisuje **n** elementów o rozmiarze **s** bajtów każdy, do pliku określanego przez **fp**, biorąc dane z obszaru pamięci wskazywanego przez **p**
- zwraca liczbę faktycznie zapisanych elementów

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

- pobiera **n** elementów o rozmiarze **s** bajtów każdy, z pliku określanego przez **fp** i umieszcza odczytane dane w obszarze pamięci wskazywanym przez **p**
- zwraca liczbę faktycznie odczytanych elementów

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);

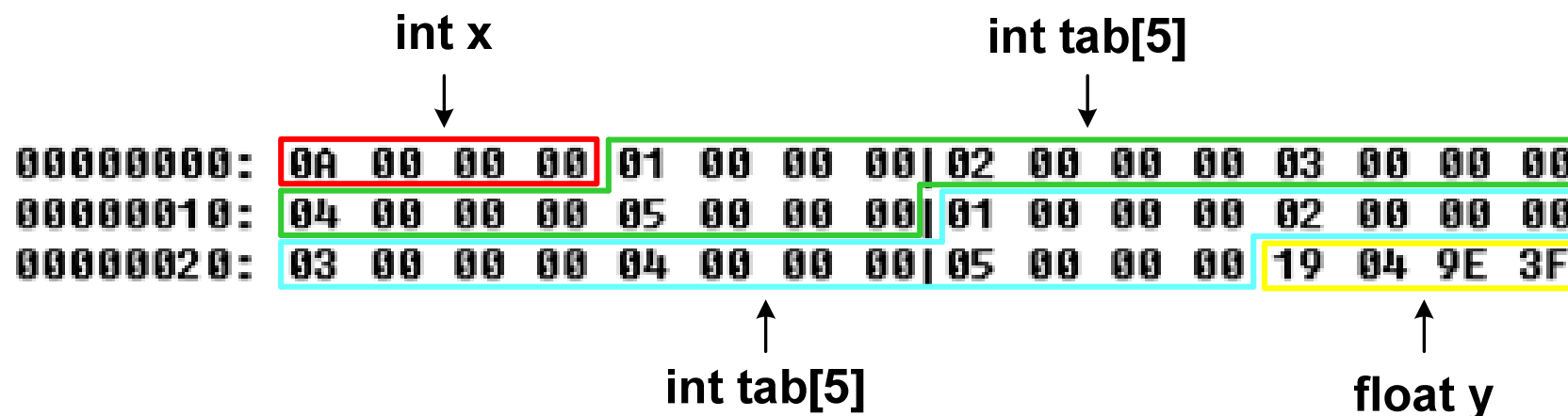
    return 0;
}
```

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:



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 | %■■■■■■■■\$■■■■■■■■
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 | %■■■■■■■■\$■■■■■■■■
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 | %■■■■■■■■\$■■■■■■■■
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

```
void rewind(FILE *fp);
```

- ustawia wskaźnik pozycji w pliku wskazywanym przez **fp** na początek pliku

```
long int ftell(FILE *fp);
```

- zwraca bieżące położeniu w pliku (liczbę bajtów od początku pliku)

```
int fseek(FILE *fp, long int offset, int mode);
```

- przechodzi bezpośrednio do dowolnego bajtu w pliku wskazywanym przez **fp**
- **offset** - wielkość przejścia w bajtach, **mode** - punkt początkowy (**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**

Rekordowe (blokowe) operacje wejścia-wyjścia

```
int fgetpos(FILE *fp, fpos_t *pos);
```

- zapamiętuję pod zmienną `pos` bieżące położenie w pliku wskazywanym przez `fp`
- zwraca `0`, gdy wywołanie jest poprawne i wartość niezerową, gdy wystąpił błąd

```
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 13

Dziękuję za uwagę!