



Wydział Elektryczny
Katedra Elektrotechniki Teoretycznej i Metrologii

Materiały do wykładu z przedmiotu:
Informatyka
Kod: **EDS1B1007**

WYKŁAD NR 9

Opracował: **dr inż. Jarosław Forenc**
Białystok 2019

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 9

- Przekazywanie argumentów do funkcji przez wartość i wskaźnik
- Operacje wejścia-wyjścia w języku C, strumienie
- Operacje na plikach
 - otwarcie pliku
 - zamknięcie pliku

Funkcje w języku C

```
#include <stdio.h> /* przekatna kwadratu */  
#include <math.h>
```

```
float przekatna(float bok) ← definicja funkcji  
{  
    float wynik;  
    wynik = bok * sqrt(2.0f);  
    return wynik;  
}
```

```
int main(void) ← definicja funkcji  
{  
    float a = 10.0f, d;  
    d = przekatna(a);  
    printf("Bok = %g, przekatna = %g\n", a, d);  
    return 0;  
}
```

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 przez wartość

```
#include <stdio.h>

void fun(int a)
{
    a = 10;
    printf("fun: a = %d\n", a);
}

int main(void)
{
    int a = 20;

    fun(a);
    printf("main: a = %d\n", a);

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	20	main()

Przekazywanie argumentów przez wartość

```
#include <stdio.h>

void fun(int a)
{
    a = 10;
    printf("fun: a = %d\n", a);
}

int main(void)
{
    int a = 20;

    fun(a);
    printf("main: a = %d\n", a);

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	20	main()
a	0x0024FAF8	20	fun()

Przekazywanie argumentów przez wartość

```
#include <stdio.h>

void fun(int a)
{
    a = 10;
    printf("fun: a = %d\n", a);
}

int main(void)
{
    int a = 20;

    fun(a);
    printf("main: a = %d\n", a);

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	20	main()
a	0x0024FAF8	10	fun()

fun: a = 10

Przekazywanie argumentów przez wartość

```
#include <stdio.h>

void fun(int a)
{
    a = 10;
    printf("fun: a = %d\n", a);
}

int main(void)
{
    int a = 20;

    fun(a);
    printf("main: a = %d\n", a);

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	20	main()

fun: a = 10
main: a = 20

Przekazywanie argumentów przez wskaźnik

```
#include <stdio.h>

void fun(int *a)
{
    *a = 10;
    printf("fun: a = %d\n", *a)
}

int main(void)
{
    int a = 20;

    fun(&a);
    printf("main: a = %d\n", a);

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	20	main()

Przekazywanie argumentów przez wskaźnik

```
#include <stdio.h>

void fun(int *a)
{
    *a = 10;
    printf("fun: a = %d\n", *a)
}

int main(void)
{
    int a = 20;

    fun(&a);
    printf("main: a = %d\n", a);

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	20	main()
a	0x0024FAF8	0x0024FBDC	fun()

Przekazywanie argumentów przez wskaźnik

```
#include <stdio.h>

void fun(int *a)
{
    *a = 10;
    printf("fun: a = %d\n", *a)
}

int main(void)
{
    int a = 20;

    fun(&a);
    printf("main: a = %d\n", a);

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	10	main()
a	0x0024FAF8	0x0024FBDC	fun()

fun: a = 10

Przekazywanie argumentów przez wskaźnik

```
#include <stdio.h>

void fun(int *a)
{
    *a = 10;
    printf("fun: a = %d\n", *a)
}

int main(void)
{
    int a = 20;

    fun(&a);
    printf("main: a = %d\n", a);

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	10	main()

fun: a = 10
main: a = 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
    }

    drukuj
    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`)

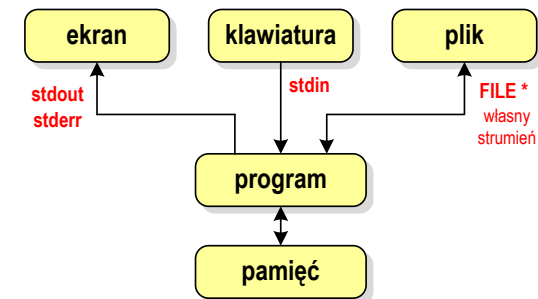
Strumienie

- 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])
```

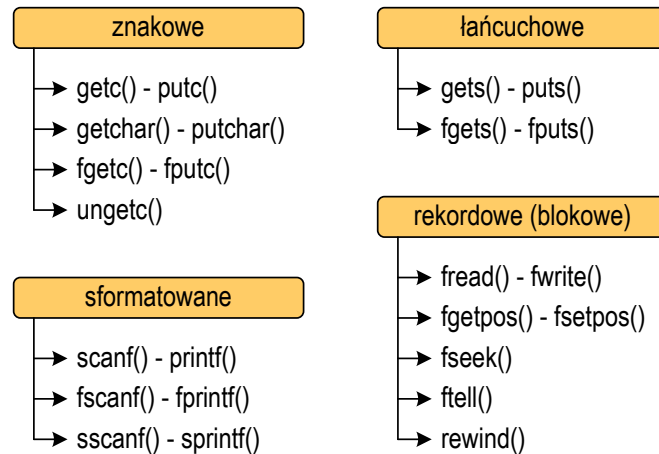
- Funkcja `printf()` niejawnie używa strumienia `stdout`
- Funkcja `scanf()` niejawnie używa strumienia `stdin`

Współpraca programu z „otoczeniem”

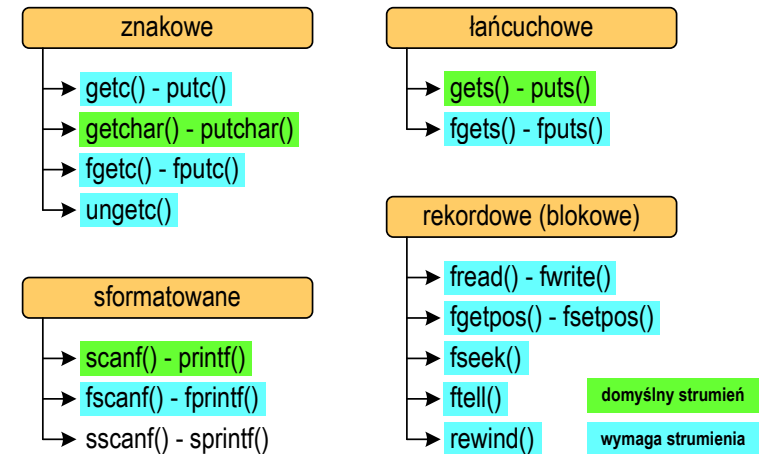


- Standardowe funkcje wejścia-wyjścia mogą:
 - domyślnie korzystać z określonego strumienia (`stdin`, `stdout`, `stderr`)
 - wymagać podania strumienia (własnego, `stdin`, `stdout`, `stderr`)

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;
}
```

Koniec wykładu nr 9

Dziękuję za uwagę!