Informatyka 1 (ES1F1002)

Politechnika Białostocka - Wydział Elektryczny

Elektrotechnika, semestr II, studia stacjonarne I stopnia

Rok akademicki 2022/2023

Wykład nr 13 (16.01.2023)

dr inż. Jarosław Forenc

Plan wykładu nr 13

- Funkcje w języku C
  - typy funkcji
  - przekazywanie argumentów do funkcji przez wartość i przez wskaźnik
  - przekazywanie wektorów, macierzy i struktur do funkcji
- Operacje wejścia-wyjścia w języku C
  - typy standardowych operacji wejścia wyjścia
  - strumienie, standardowe strumienie: stdin, stdout, stderr
- Operacje na plikach
  - otwarcie i zamknięcie pliku
  - operacje znakowe, łańcuchowe
  - operacje sformatowane

Funkcje w języku C

```
#include <stdio.h>     /* przekątna 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;
}
```


Typy funkcji (1)

- Dotychczas prezentowane funkcje miały argumenty i zwracały wartości
- Struktura i wywołanie takiej funkcji ma następującą postać

```
typ nazwa(parametry)
{
   instrukcje;
   return wartość;
}
```

```
typ zm;
zm = nazwa(argumenty);
```

- Można zdefiniować także funkcje, które nie mają argumentów i/lub nie zwracają żadnej wartości

Typy funkcji (2)

- Funkcja bez argumentów i nie zwracająca wartości:
  - w nagłówku funkcji, typ zwracanej wartości to void
  - zamiast parametrów, podaje się słowo void lub nie wpisuje się nic
  - jeśli występuje return, to nie może po nim znajdować się żadna wartość
  - jeśli return nie występuje, to funkcja kończy się po wykonaniu wszystkich instrukcji
- Struktura funkcji:

```
void nazwa(void)
{
   instrukcje;
   return;
}
```

```
void nazwa()
{
   instrukcje;
   return;
}
```


Typy funkcji (2)

- Funkcja bez argumentów i nie zwracająca wartości:
  - w nagłówku funkcji, typ zwracanej wartości to void
  - zamiast parametrów, podaje się słowo void lub nie wpisuje się nic
  - jeśli występuje return, to nie może po nim znajdować się żadna wartość
  - jeśli return nie występuje, to funkcja kończy się po wykonaniu wszystkich instrukcji
- Struktura funkcji:

```
void nazwa(void)
{
   instrukcje;
}
```

```
void nazwa()
{
   instrukcje;
}
```

- Wywołanie funkcji:

```
nazwa();
```


Typy funkcji (2) - przykład

```
#include <stdio.h>

void drukuj_linie(void)
{
    printf("--------------------------\n");
}

int main(void)
{
    drukuj_linie();
    printf("Funkcje nie sa trudne!\n");
    drukuj_linie();

    return 0;
}
```

```
--------------------------
Funkcje nie sa trudne!
--------------------------
```


Typy funkcji (3)

- Funkcja z argumentami i nie zwracająca wartości:
  - w nagłówku funkcji, typ zwracanej wartości to void
  - jeśli występuje return, to nie może po nim znajdować się żadna wartość
  - jeśli return nie występuje, to funkcja kończy się po wykonaniu wszystkich instrukcji
- Struktura funkcji:

```
void nazwa(parametry)
{
   instrukcje;
   return;
}
```

```
void nazwa(parametry)
{
   instrukcje;
}
```

- Wywołanie funkcji:

```
nazwa(argumenty);
```

Typy funkcji (3) - przykład

```
#include <stdio.h>

void drukuj_dane(char *imie, char *nazwisko, int wiek)
{
    printf("Imie:           %s\n",imie);
    printf("Nazwisko:       %s\n",nazwisko);
    printf("Wiek:           %d\n",wiek);
    printf("Rok urodzenia:  %d\n\n",2023-wiek);
}

int main(void)
{
    drukuj_dane("Jan","Kowalski",23);
    drukuj_dane("Barbara","Nowak",28);

    return 0;
}
```


Typy funkcji (3) - przykład

```
#include <stdio.h>

void drukuj_dane(char *imie,
{
    printf("Imie:
    printf("Nazwisko:
    printf("Wiek:
    printf("Rok urodzenia:
}

int main(void)
{
    drukuj_dane("Jan","Kowalski",24);
    drukuj_dane("Barbara","Nowak",29);

    return 0;
}
```

```
Imie:           Jan
Nazwisko:       Kowalski
Wiek:           24
Rok urodzenia:  1999

Imie:           Barbara
Nazwisko:       Nowak
Wiek:           29
Rok urodzenia:  1994
```


Typy funkcji (4)

- Funkcja bez argumentów i zwracająca wartość:
  - zamiast parametrów, podaje się słowo void lub nie wpisuje się nic
  - typ zwracanej wartości musi być zgodny z typem w nagłówku funkcji
- Struktura funkcji:

```
typ nazwa(void)
{
   instrukcje;
   return wartość;
}
```

```
typ nazwa()
{
   instrukcje;
   return wartość;
}
```

- Wywołanie funkcji:

```
typ zm;
zm = nazwa();
```


Typy funkcji (4) - przykład

```
W roku jest: 31536000 sekund
```

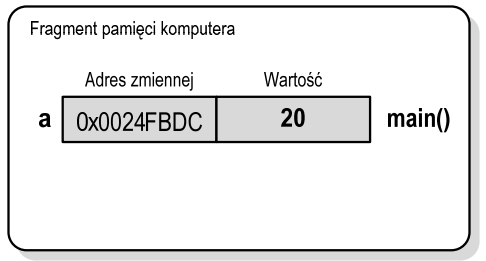
```
#include <stdio.h>

int liczba_sekund_rok(void)
{
    return (365 * 24 * 60 * 60);
}

int main(void)
{
    int wynik;

    wynik = liczba_sekund_rok();
    printf("W roku jest: %d sekund\n",wynik);

    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

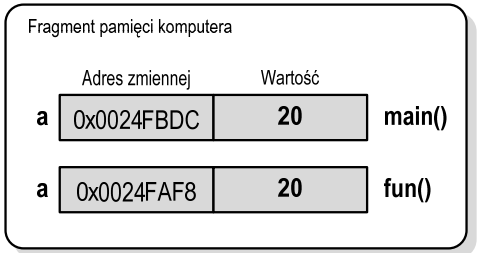
Przekazywanie argumentów przez wartość

```
#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;
}
```



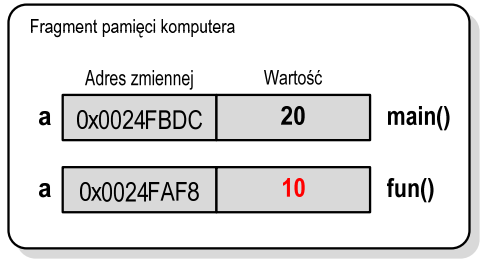
Przekazywanie argumentów przez wartość

```
#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;
}
```



Przekazywanie argumentów przez wartość

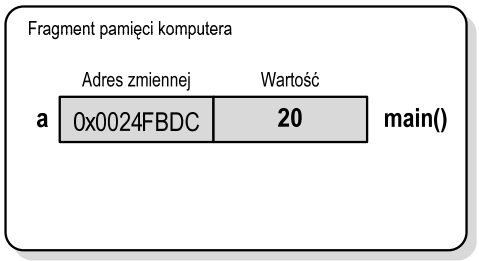
```
#include <stdio.h>

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;
}
```



fun: a = 10

Przekazywanie argumentów przez wartość

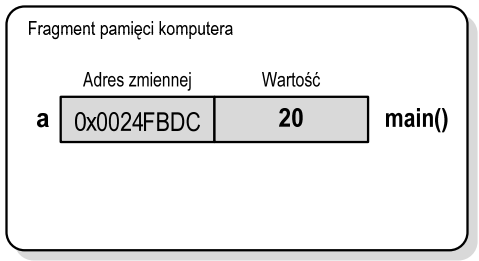
```
#include <stdio.h>

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;
}
```



fun: a = 10
main: a = 20

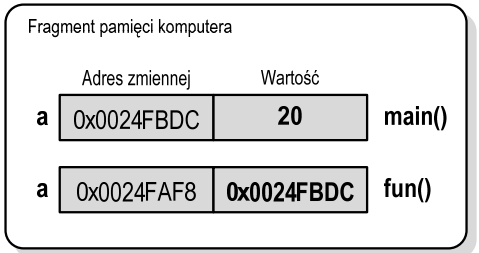
Przekazywanie argumentów przez wskaźnik

```
#include <stdio.h>

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

int main(void)
{
  int a = 20;

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

  return 0;
}
```



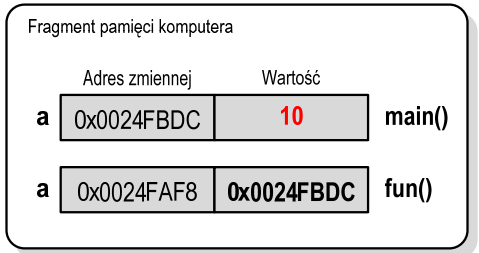
Przekazywanie argumentów przez wskaźnik

```
#include <stdio.h>

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

int main(void)
{
  int a = 20;

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

  return 0;
}
```



Przekazywanie argumentów przez wskaźnik

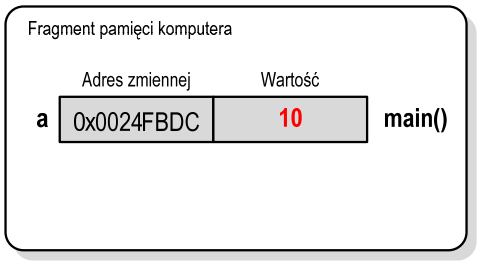
```
#include <stdio.h>

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

int main(void)
{
  int a = 20;

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

  return 0;
}
```



fun: a = 10

Przekazywanie argumentów przez wskaźnik

```
#include <stdio.h>

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

int main(void)
{
  int a = 20;

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

  return 0;
}
```




```
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
       {

   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
- 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

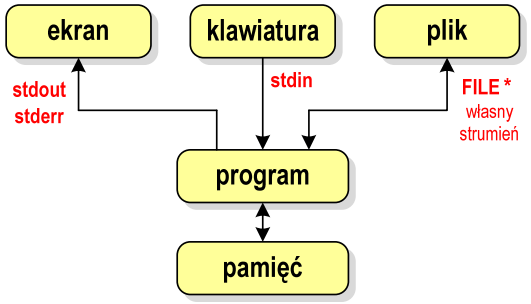
- W każdym programie automatycznie tworzone są i otwierane trzy standardowe strumienie wejścia-wyjścia:
  - stdin - standardowe wejście, skojarzone z klawiaturą
  - stdout - standardowe wyjście , skojarzone z ekranem monitora
  - stderr - standardowe wyjście dla komunikatów o błędach, skojarzone z ekranem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

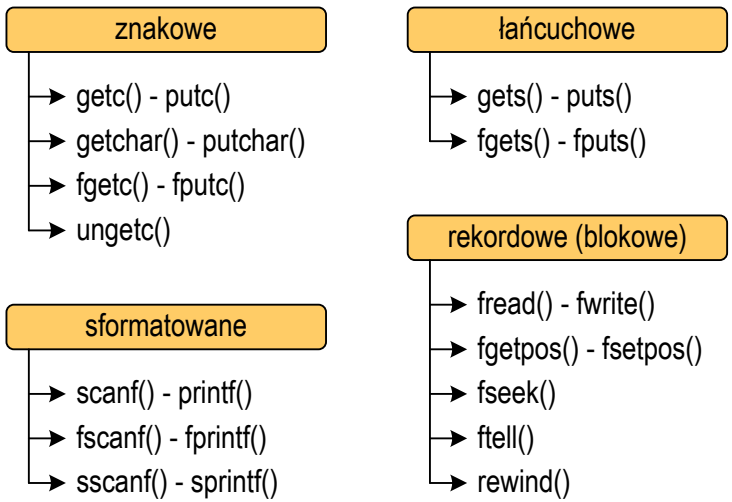
Strumienie

- Współpraca programu z „otoczeniem”

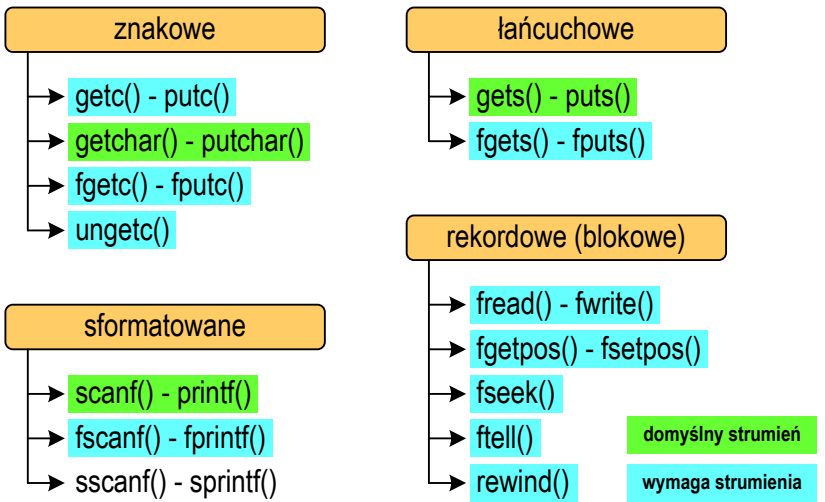


- Standardowe funkcje wejścia-wyjścia mogą:
  - domyślnie korzystać z określonego strumienia (stdin, stdout, stderr)
  - wymagać podania strumienia (własnego, stdin, stdout, stderr)

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()

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);
```

- 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

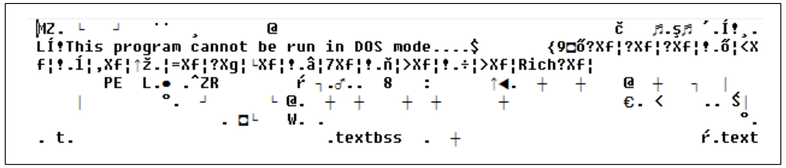
```
#include <stdio.h>

int main(void)
{
    FILE *fp;

    fp = fopen("plik.txt","w");
    if (fp == NULL)
    {
        printf("Blad 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):



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■
```

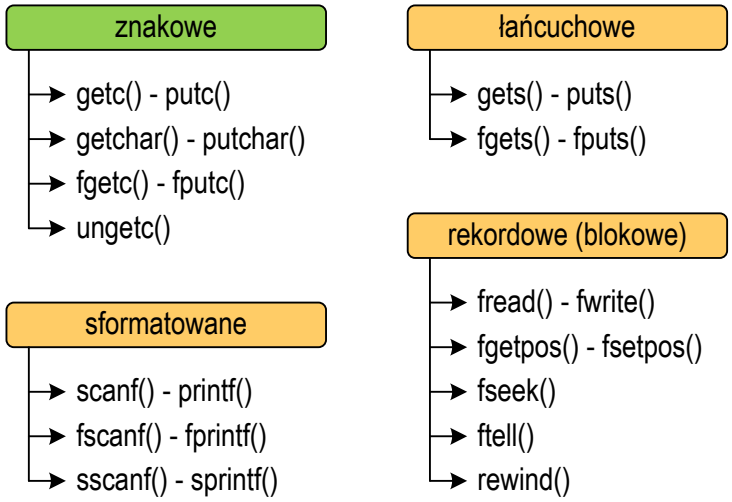
- Pliki binarne nie mają ściśle określonej struktury

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

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

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

odczytanie danych z pliku
sprawdzenie czy koniec pliku
przetworzenie danych
odczytanie 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

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

```
abcdefghijklmnopqrstuvwxyz
```


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

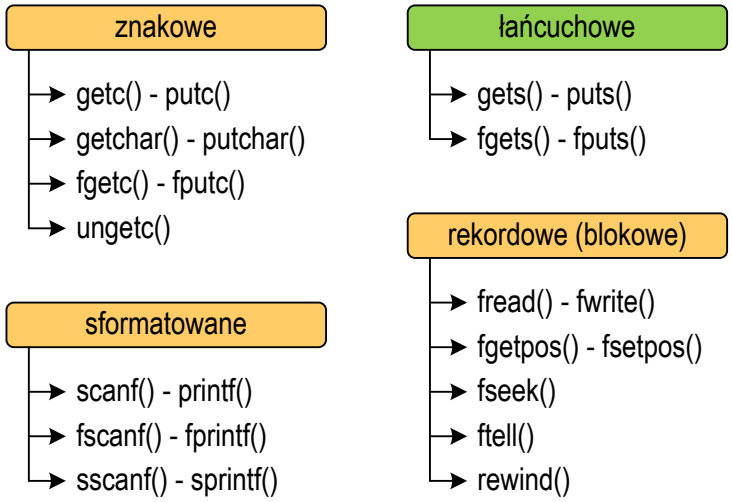
Przykład: liczba wyrazów w pliku

```
#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 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

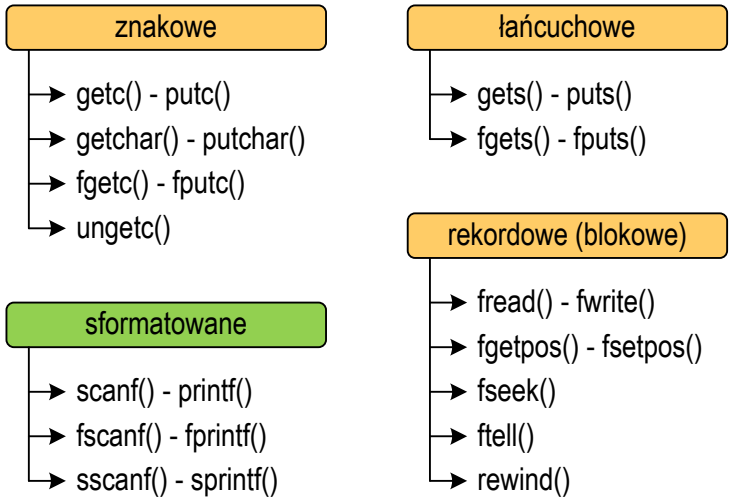
- Kolejne wywołania funkcji fgets(buf,15,fp); i zawartość tablicy buf

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

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

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]
```


Koniec wykładu nr 13

Dziękuję za uwagę!