Wydział Elektryczny
Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki

Materiały do wykładu z przedmiotu:
Informatyka
Kod: EDS1B1007

WYKŁAD NR 6

Opracował: dr inż. Jarosław Forenc

Białystok 2022

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 6

- Funkcje w języku C
  - ogólna struktura, argumenty i parametry funkcji
  - domyślne wartości parametrów funkcji
  - wskaźniki do funkcji, wywołanie funkcji przez wskaźnik
  - prototypy funkcji, 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

Program w języku C

- Program w języku C składa się z funkcji i zmiennych
  - funkcje zawierają instrukcje wykonujące operacje
  - zmienne przechowują wartości

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

int main(void)
{
    float a = 10.0f, d;

    d = a * sqrt(2.0f);
    printf("Bok = %g, przekatna = %g\n",a,d);

    return 0;
}
```

```
Bok = 10, przekatna = 14.1421
```

Program w języku C

- Program w języku C składa się z funkcji i zmiennych
  - funkcje zawierają instrukcje wykonujące operacje
  - zmienne przechowują wartości

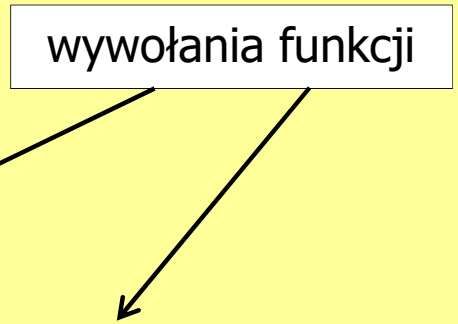
```
#include <stdio.h>     /* przekątna kwadratu */
#include <math.h>

int main(void)
{
    float a = 10.0f, d;

    d = a * sqrt(2.0f);
    printf("Bok = %g, przekatna = %g\n",a,d);

    return 0;
}
```

definicja funkcji

Program w języku C

- Program w języku C składa się z funkcji i zmiennych
  - funkcje zawierają instrukcje wykonujące operacje
  - zmienne przechowują wartości

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

int main(void)
{
    float a = 10.0f, d;

    d = a * sqrt(2.0f);
    printf("Bok = %g, przekatna = %g\n",a,d);

    return 0;
}
```



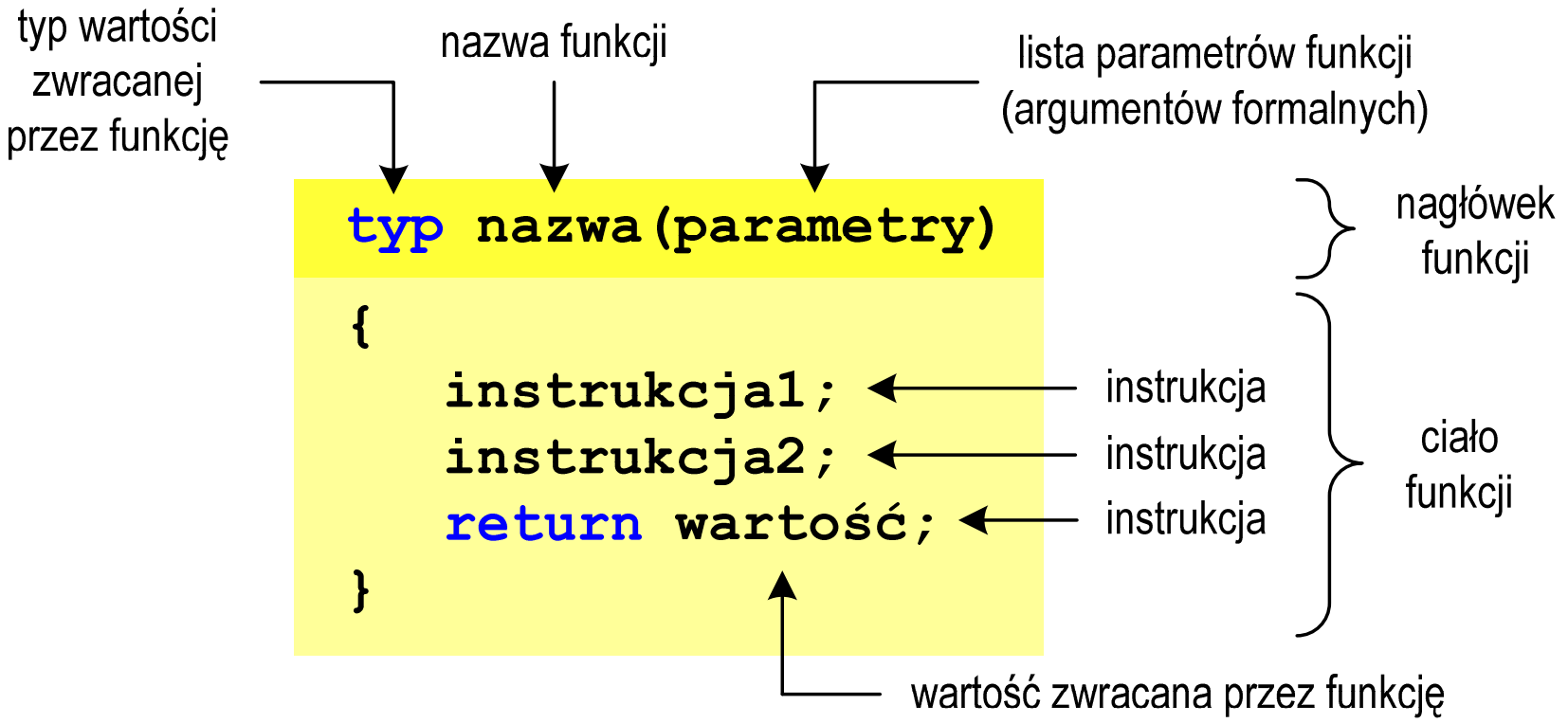
Funkcje w języku C

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

float przekatna(float bok)
{
    float wynik;
    wynik = bok * sqrt(2.0f);
    return wynik;
}

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

definicja funkcji

definicja funkcji

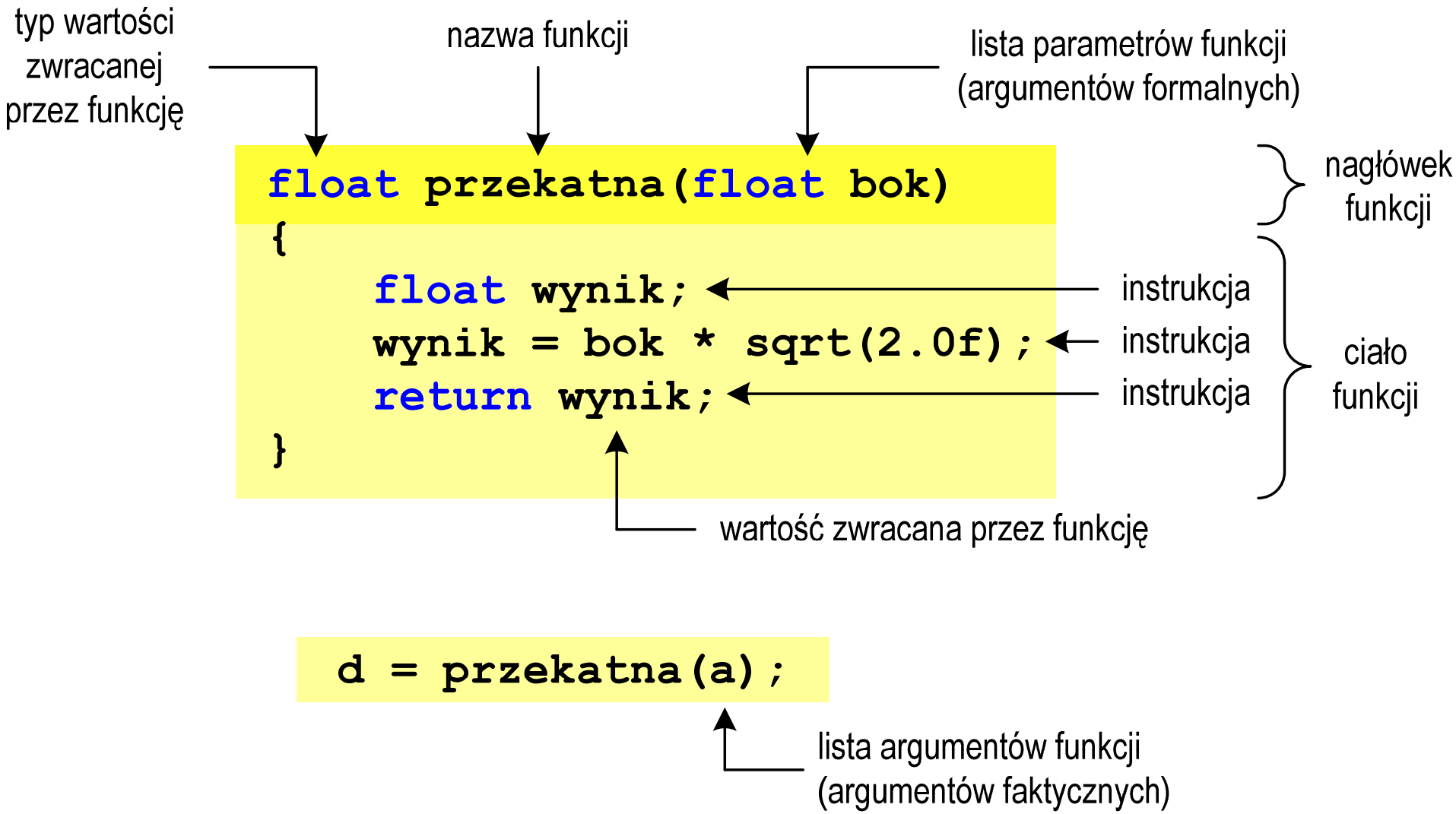
Ogólna struktura funkcji w języku C




```
zmienna = nazwa(argumenty);
```

lista argumentów funkcji (argumentów faktycznych)

Ogólna struktura funkcji w języku C




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

```
d = przekatna(a);
```

Argumenty funkcji

- Argumentami funkcji mogą być stałe liczbowe, zmienne, wyrażenia arytmetyczne, wywołania innych funkcji

```
d = przekatna(a);
d = przekatna(10);
d = przekatna(2*a+5);
d = przekatna(sqrt(a)+15);
```

- Wywołanie funkcji może być argumentem innej funkcji

```
printf("Bok = %g, przekatna = %g\n",
                                a, przekatna(a));
```

Parametry funkcji

- Parametry funkcji traktowane są tak samo jak zmienne zadeklarowane w tej funkcji i zainicjalizowane wartościami argumentów wywołania

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

- Funkcję przekatna() można zapisać w prostszej postaci:

```
float przekatna(float bok)
{
    return bok * sqrt(2.0f);
}
```

Parametry funkcji

- Jeśli funkcja ma kilka parametrów, to dla każdego z nich podaje się:
  - typ parametru
  - nazwę parametru
- Parametry oddzielane są od siebie przecinkami

```
/* przekątna prostokąta */

float przekatna(float a, float b)
{
    return sqrt(a*a+b*b);
}
```

Parametry funkcji

- W różnych funkcjach zmienne mogą mieć takie same nazwy

```
#include <stdio.h>     /* przekątna prostokąta */
#include <math.h>

float przekatna(float a, float b)
{
    return sqrt(a*a+b*b);
}

int main(void)
{
    float a = 10.0f, b = 5.5f, d;
    d = przekatna(a,b);
    printf("Przekatna prostokata = %g\n",d);
    return 0;
}
```

Domyślne wartości parametrów funkcji

- W definicji funkcji można jej parametrom nadać domyślne wartości

```
float przekatna(float a = 10, float b = 5.5f)
{
    return sqrt(a*a+b*b);
}
```

- W takim przypadku funkcję można wywołać z dwoma, jednym lub bez żadnych argumentów

```
d = przekatna(a,b);
```

```
d = przekatna(a);
```

```
d = przekatna();
```

- Brakujące argumenty zostaną zastąpione wartościami domyślnymi

Domyślne wartości parametrów funkcji

- Nie wszystkie parametry muszą mieć podane domyślne wartości
- Wartości muszą być podawane od prawej strony listy parametrów

```
float przekatna(float a, float b = 5.5f)
{
    return sqrt(a*a+b*b);
}
```

- Powyższa funkcja może być wywołana z jednym lub dwoma argumentami

```
d = przekatna(a,b);
```

```
d = przekatna(a);
```

- Domyślne wartości parametrów mogą być podane w deklaracji lub w definicji funkcji

Wartość zwracana przez funkcję

- Słowo kluczowe return może wystąpić w funkcji wiele razy

```
float ocena(int pkt)
{
    if (pkt>90)              return 5.0f;
    if (pkt>80 && pkt<91)    return 4.5f;
    if (pkt>70 && pkt<81)    return 4.0f;
    if (pkt>60 && pkt<71)    return 3.5f;
    if (pkt>50 && pkt<61)    return 3.0f;
    if (pkt<51)              return 2.0f;
}
```

91-100 pkt. → 5,0	81-90 pkt. → 4,5
71-80 pkt. → 4,0	61-70 pkt. → 3,5
51-60 pkt. → 3,0	0-50 pkt. → 2,0

Wskaźniki do funkcji

- Definicja funkcji

```
typ nazwa_funkcji(parametry)
{
}
```

- Można deklarować wskaźniki do funkcji

```
typ (*nazwa_wskaźnika)(parametry);
```

- Przykłady deklaracji funkcji i odpowiadającym im wskaźników

```
void foo();
int foo(double x);
void foo(char *x);
int *foo(int x,int y);
float *foo(void);
```

```
void (*fptr)();
int (*fptr)(double);
void (*fptr)(char *);
int *(*fptr)(int,int);
float *(*fptr)(void);
```

Wywołanie funkcji przez wskaźnik

```
#include <stdio.h>

int suma(int x, int y)
{
    return x + y;
}

int main(void)
{
    int (*fptr)(int,int);     // deklaracja wskaźnika do funkcji
    int w;

    fptr = suma;              // przypisanie wskaźnikowi adresu funkcji
    w = fptr(5,10);           // wywołanie funkcji przez wskaźnik
    printf("w = %d\n",w);

    return 0;
}
```

```
w = 15
```

Prototyp funkcji

- Czy można zmienić kolejność definicji funkcji w kodzie programu?

```
#include <stdio.h>     /* przekątna prostokąta */
#include <math.h>

float przekatna(float a, float b)
{
    return sqrt(a*a+b*b);
}

int main(void)
{
    float a = 10.0f, b = 5.5f, d;
    d = przekatna(a,b);
    printf("Przekatna prostokata = %g\n",d);
    return 0;
}
```

definicja funkcji

definicja funkcji

Prototyp funkcji

- Czy można zmienić kolejność definicji funkcji w kodzie programu?

```
#include <stdio.h>     /* przekątna prostokąta */
#include <math.h>

int main(void)                                          definicja funkcji
{
    float a = 10.0f, b = 5.5f, d;
    d = przekatna(a,b);
    printf("Przekatna prostokata = %g\n",d);
    return 0;
}

float przekatna(float a, float b)                       definicja funkcji
{
    return sqrt(a*a+b*b);
}
```

Prototyp funkcji

- Czy można zmienić kolejność definicji funkcji w kodzie programu?

```
#include <stdio.h>     /* przekątna prostokąta */
#include <math.h>

int main(void)
{
    float a = 10.0f, b = 5.5f, d;
    d = przekatna(a,b);
    printf("Przekatna prostokata = %g\n",d);
    return 0;
}

float przekatna(float a, float b)
{
    return sqrt(a*a+b*b);
}
```

definicja funkcji

error C3861: 'przekatna': identifier not found

Prototyp funkcji

```
#include <stdio.h>     /* przekątna prostokąta */
#include <math.h>

float przekatna(float a, float b);
```

prototyp funkcji

```
int main(void)
{
    float a = 10.0f, b = 5.5f, d;
    d = przekatna(a,b);
    printf("Przekatna prostokata = %g\n",d);
    return 0;
}
```

definicja funkcji

```
float przekatna(float a, float b)
{
    return sqrt(a*a+b*b);
}
```

definicja funkcji

Prototyp funkcji

- Prototyp funkcji jest to jej nagłówek zakończony średnikiem

```
float przekatna(float a, float b);
```

- Inne określenia prototypu funkcji:
  - deklaracja funkcji
  - zapowiedź funkcji
- Dzięki prototypowi kompilator sprawdza w wywołaniu funkcji:
  - nazwę funkcji
  - liczbę i typ argumentów
  - typ zwracanej wartości

```
d = przekatna(a,b);
```

- Nazwy parametrów nie mają znaczenia i mogą być pominięte:

```
float przekatna(float, float);
```

Prototyp funkcji

- W przypadku umieszczenia prototypu funkcji i pominięcia jej definicji błąd wystąpi nie na etapie kompilacji, ale łączenia (linkowania)

```
#include <stdio.h>     /* przekątna prostokąta */
#include <math.h>

float przekatna(float a, float b);
```

prototyp funkcji

```
int main(void)
{
    float a = 10.0f, b = 5.5f, d;
    d = przekatna(a,b);
    printf("Przekatna prostokata = %g\n",d);
    return 0;
}
```

definicja funkcji

Prototyp funkcji

- W przypadku umieszczenia prototypu funkcji i pominięcia jej definicji błąd wystąpi nie na etapie kompilacji, ale łączenia (linkowania)

```
1>Compiling...
1>test.cpp
1>Compiling manifest to resources...
1>Microsoft (R) Windows (R) Resource Compiler Version 6.0.5724.0
1>Copyright (C) Microsoft Corporation.  All rights reserved.
1>Linking...
1>test.obj : error LNK2019: unresolved external symbol "float __cdecl
przekatna(float,float)" (?przekatna@@YAMMM@Z) referenced in function _main
1>D:\test\Debug\test.exe : fatal error LNK1120: 1 unresolved externals
```

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",2022-wiek);
}

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

    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:     1998

Imie:              Barbara
Nazwisko:          Nowak
Wiek:              29
Rok urodzenia:     1993
```

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
```

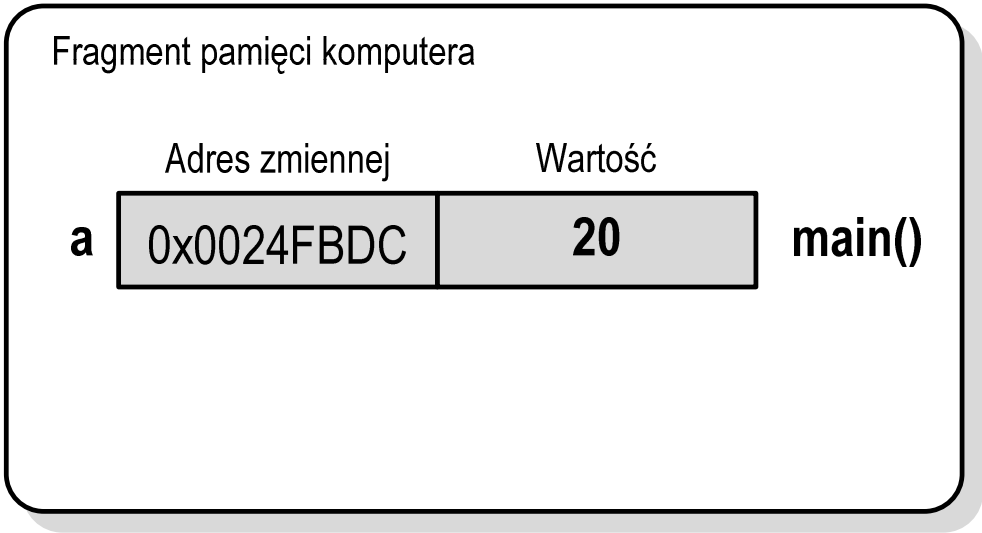
```c
#include <stdio.h>

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

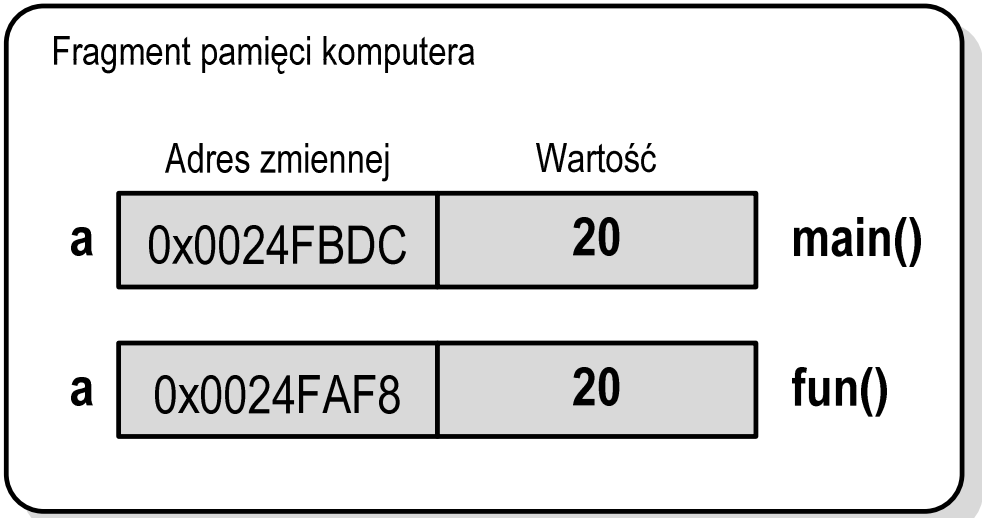
Przekazywanie argumentów przez wartość

```
#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;
}
```



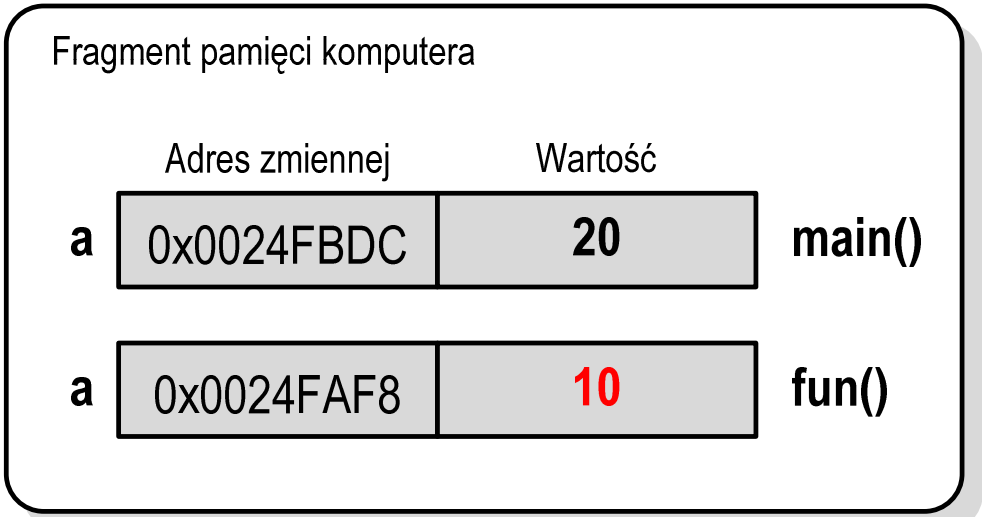
Przekazywanie argumentów przez wartość

```
#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

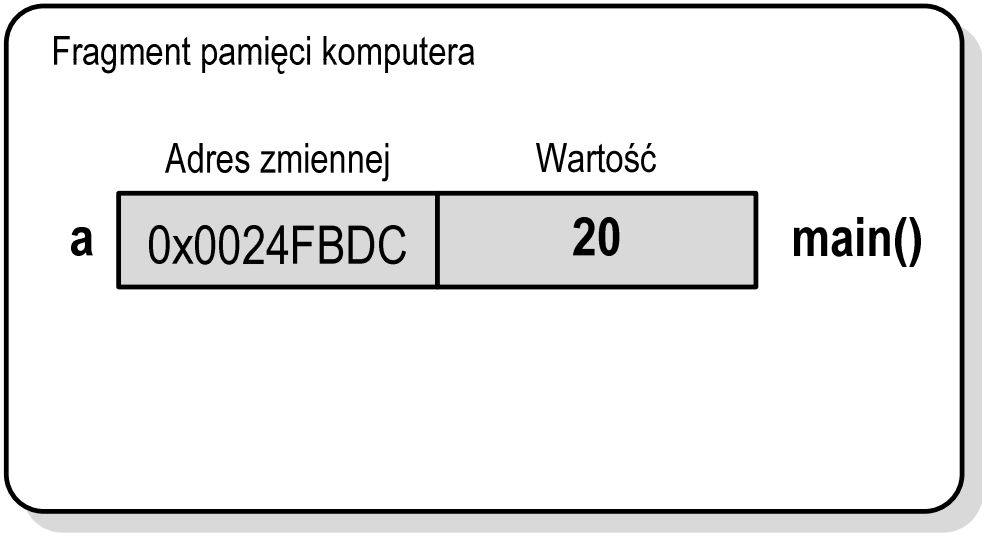
```
#include <stdio.h>

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

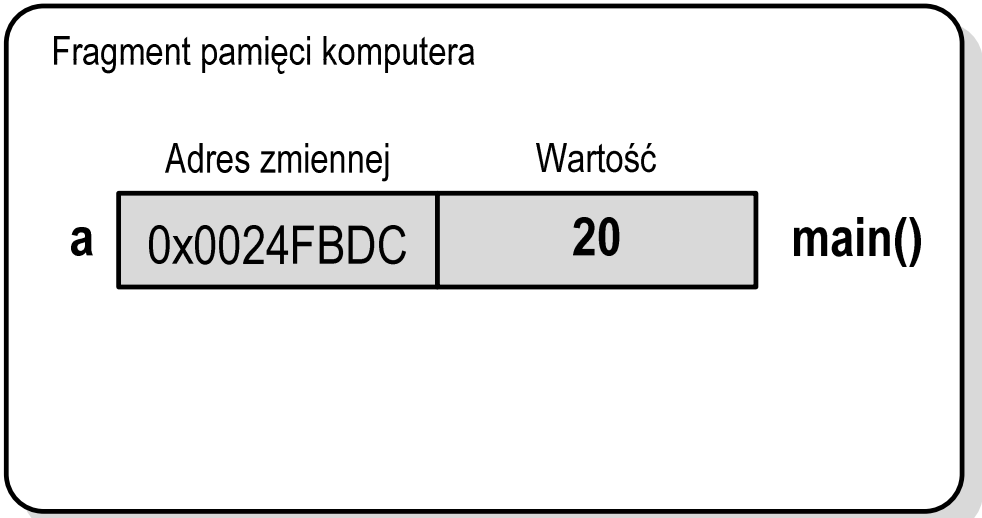
```
#include <stdio.h>

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
```

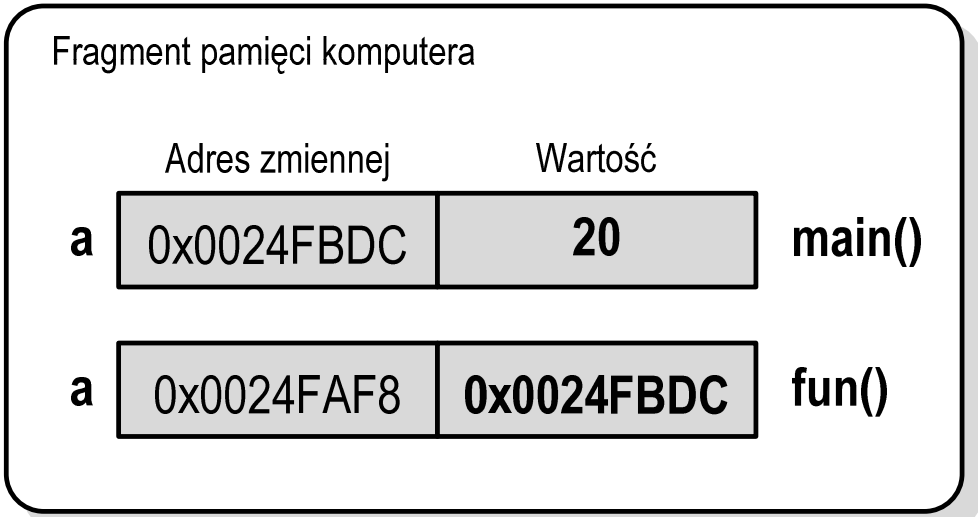
Przekazywanie argumentów przez wskaźnik

```
#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;
}
```



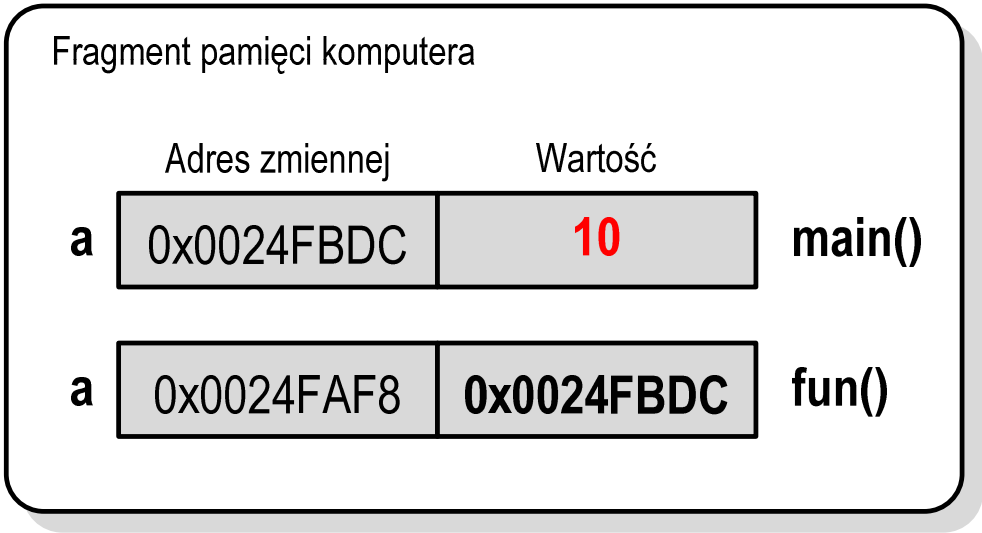
Przekazywanie argumentów przez wskaźnik

```
#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

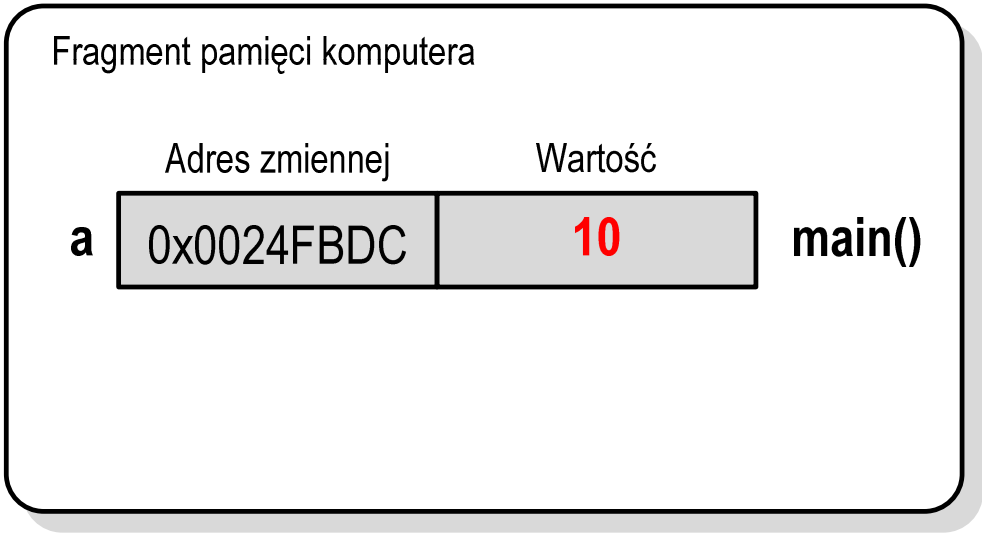
```
#include <stdio.h>

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 - const

- Jeśli funkcja nie powinna zmieniać wartości przekazywanych do niej zmiennych, to w nagłówku, przed odpowiednim parametrem, dodaje się identyfikator const

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

- Próba zmiany wartości takiego parametru powoduje błąd kompilacji

error C3892: 'tab' : you cannot assign to a variable that is const

Parametry funkcji - const

- Przykładowe prototypy funkcji z pliku nagłówkowego string.h

```
char* strcpy(char *dest, const char *source);
```

```
size_t strlen(const char *str);
```

```
char* strupr(char *str);
```

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)

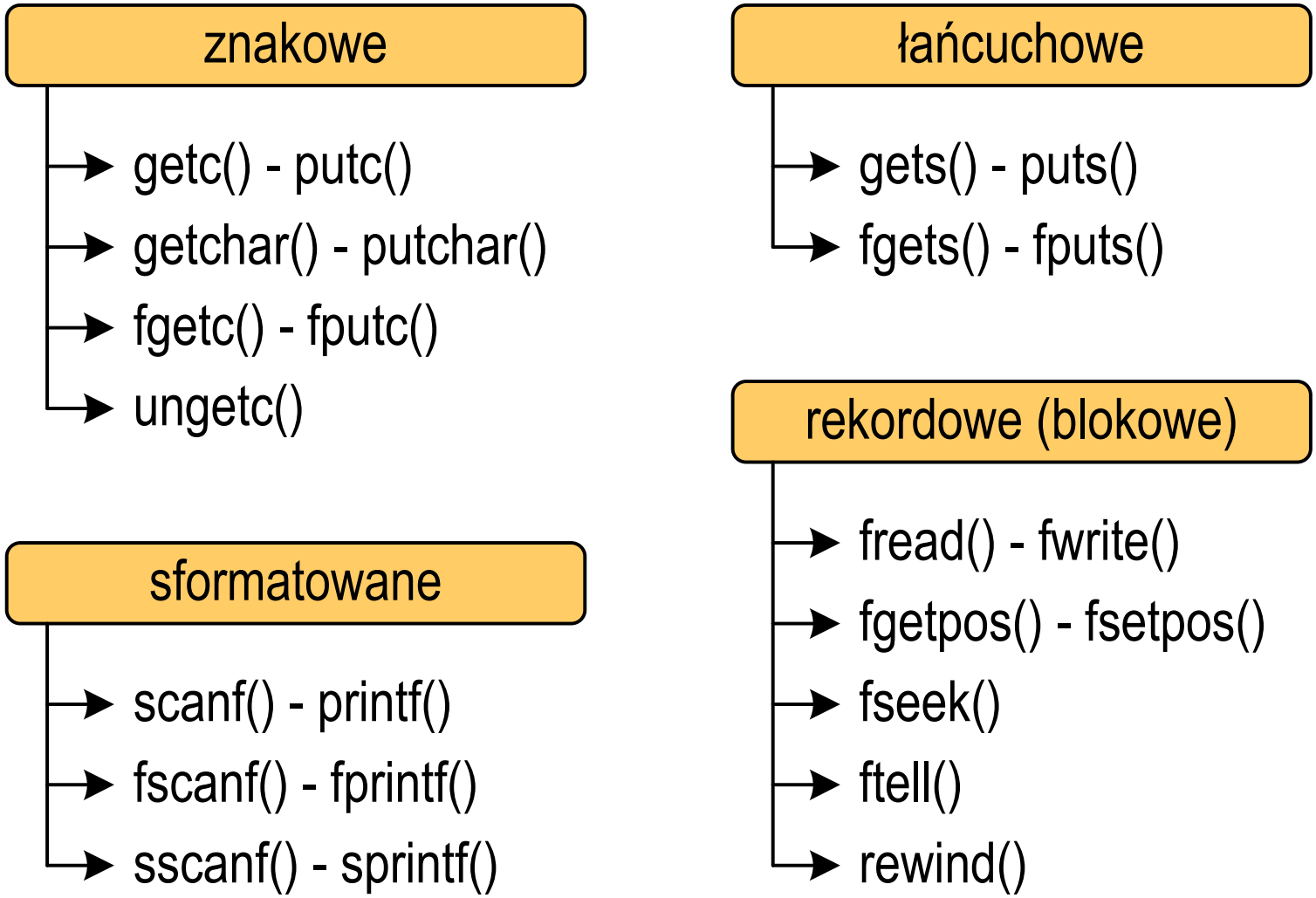
```
int main(void)
{
    struct pkt p1 = {2,3};
    struct pkt p2 = {-2,1};
    float wynik;

    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

Typy standardowych operacji wejścia-wyjścia



Strumienie

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

Strumienie

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

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

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

Strumienie

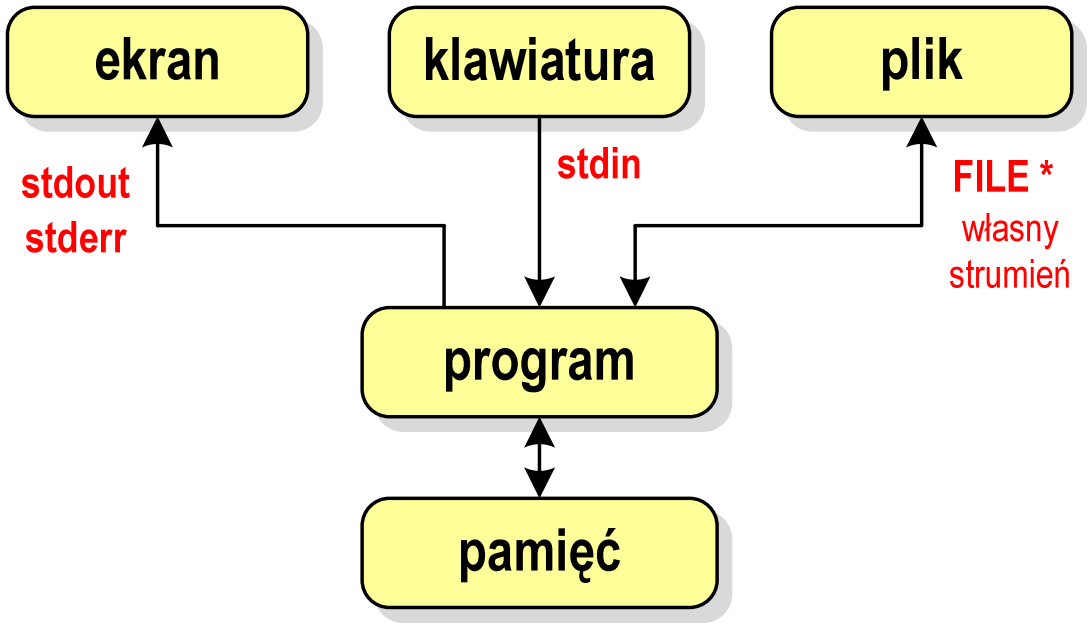
- W każdym programie automatycznie tworzone są i otwierane trzy standardowe strumienie wejścia-wyjścia:
  - stdin - standardowe wejście, skojarzone z klawiaturą
  - stdout - standardowe wyjście , skojarzone z ekranem monitora
  - stderr - standardowe wyjście dla komunikatów o błędach, skojarzone z ekranem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

```
_CRTIMP FILE * __cdecl __iob_func(void);

#define stdin  (&__iob_func()[0])
#define stdout (&__iob_func()[1])
#define stderr (&__iob_func()[2])
```

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

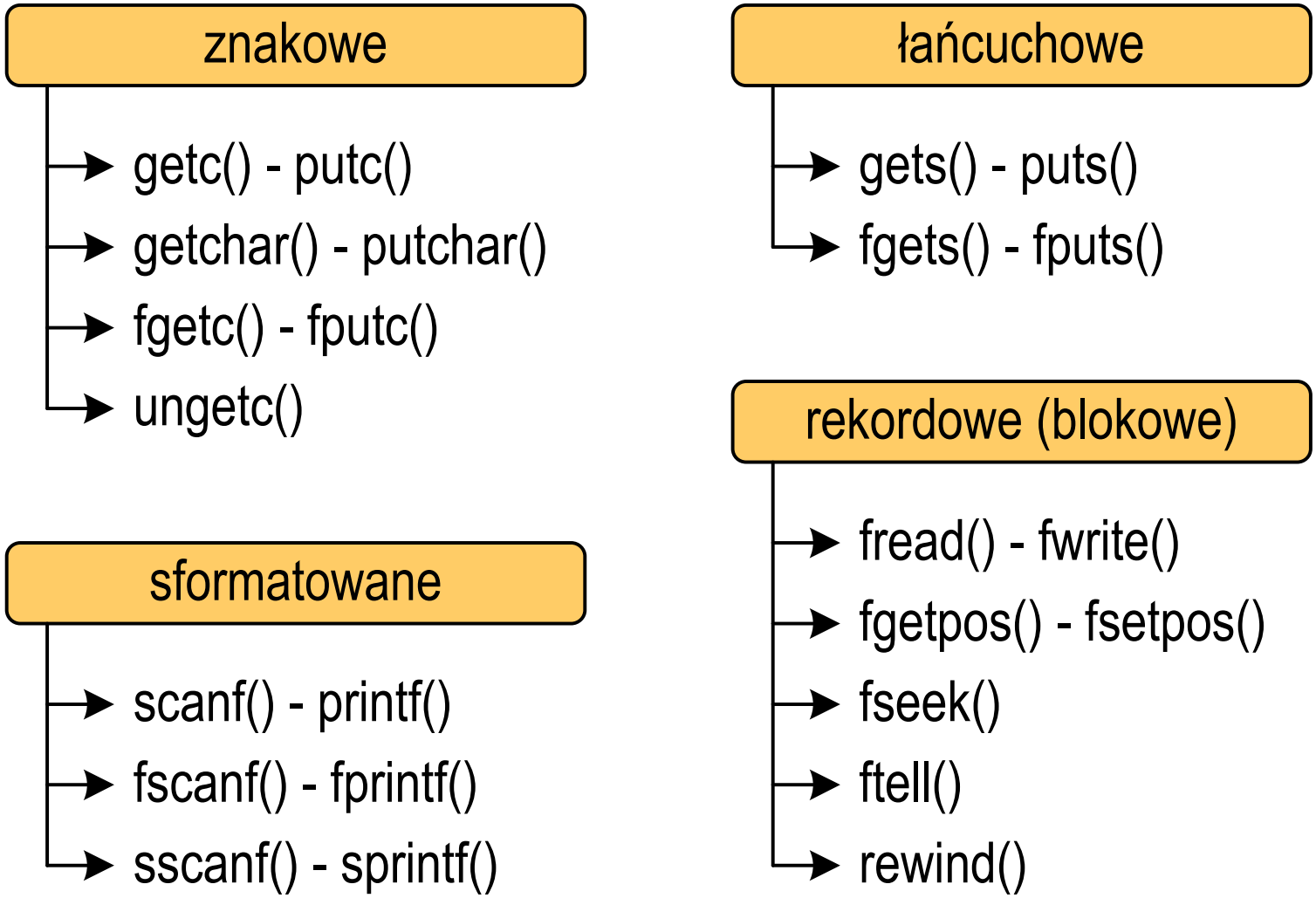
Strumienie

- Współpraca programu z „otoczeniem”

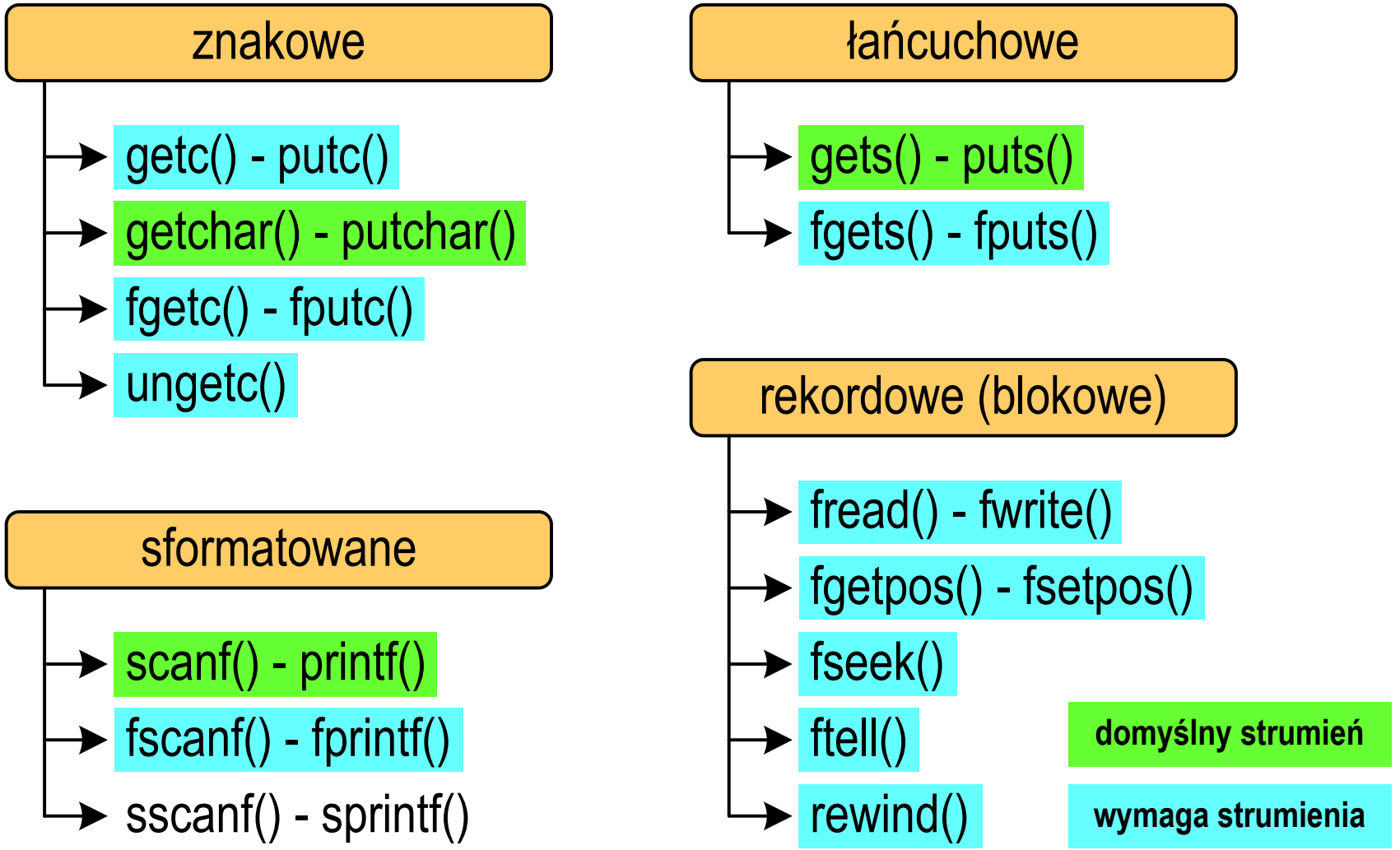


- Standardowe funkcje wejścia-wyjścia mogą:
  - domyślnie korzystać z określonego strumienia (stdin, stdout, stderr)
  - wymagać podania strumienia (własnego, stdin, stdout, stderr)

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);
```

- 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

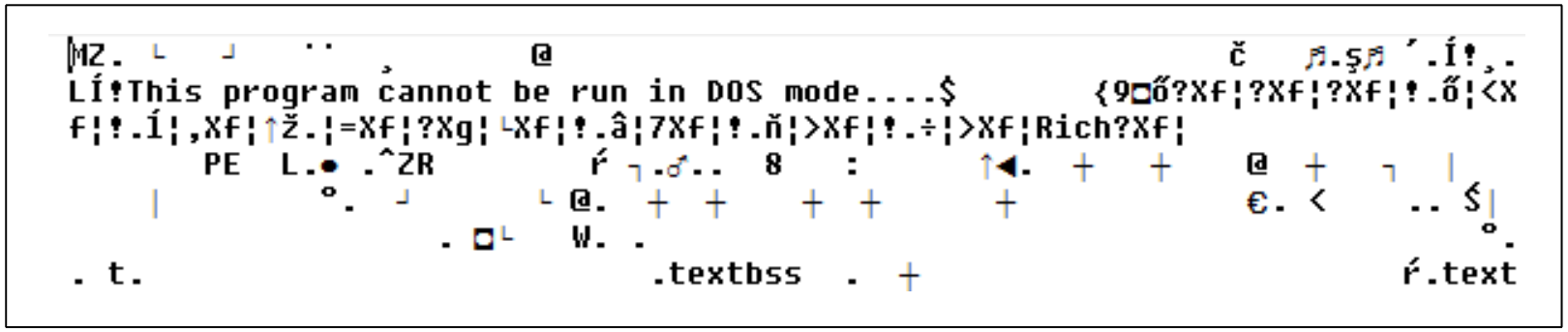
```c
#include <stdio.h>

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):

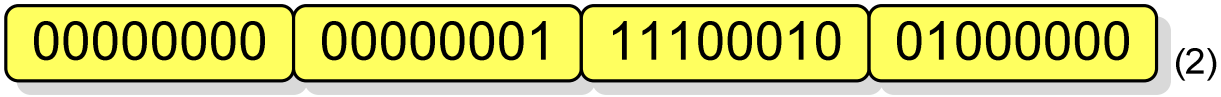


Format (plik) tekstowy i binarny

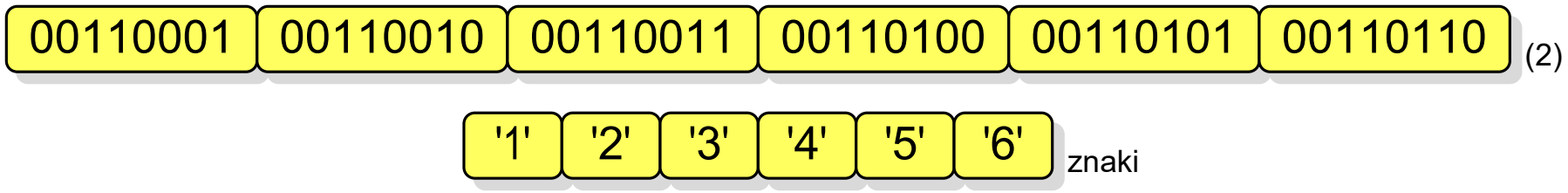
- Dane w pliku tekstowym zapisane są w postaci kodów ASCII
- Deklaracja i inicjalizacja zmiennej x typu int:

```
int x = 123456;
```

- W pamięci komputera zmienna x zajmuje 4 bajty:



- Po zapisaniu wartości zmiennej x do pliku tekstowego znajdzie się w nim 6 bajtów zawierających kody ASCII kolejnych cyfr

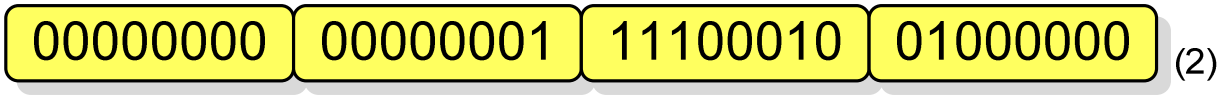


Format (plik) tekstowy i binarny

- 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:



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

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

Koniec wykładu nr 6

Dziękuję za uwagę!