Informatyka 1 (EZ1F1002)

Politechnika Białostocka - Wydział Elektryczny

Elektrotechnika, semestr I, studia niestacjonarne I stopnia

Rok akademicki 2023/2024

Wykład nr 8 (16.12.2023)

dr inż. Jarosław Forenc

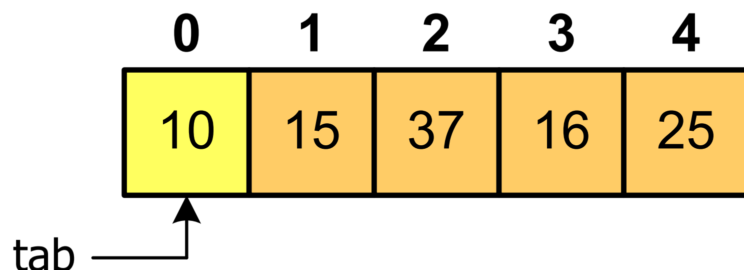
Plan wykładu nr 8

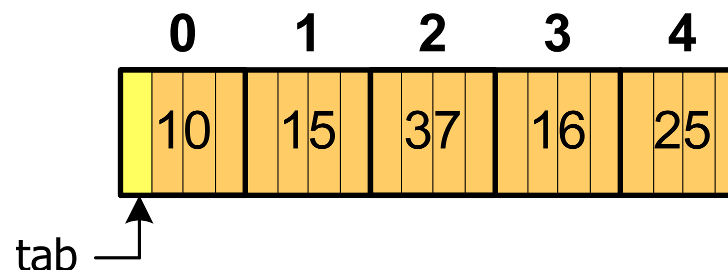
- Wskaźniki
  - związek z tablicami, dynamiczny przydział pamięci
- Funkcje w języku C
  - ogólna struktura funkcji, argumenty i parametry funkcji
  - prototypy funkcji, typy funkcji
  - przekazywanie argumentów do funkcji
- Operacje wejścia-wyjścia w języku C, pliki
  - strumienie (stdin, stdout, stderr)
  - typy standardowych operacji wejścia-wyjścia
  - otwarcie i zamknięcie pliku
  - format (plik) tekstowy i binarny

Wskaźniki a tablice

- Nazwa tablicy jest jej adresem (dokładniej - adresem elementu o indeksie 0)

```
int tab[5] = {10,15,37,16,25};
```



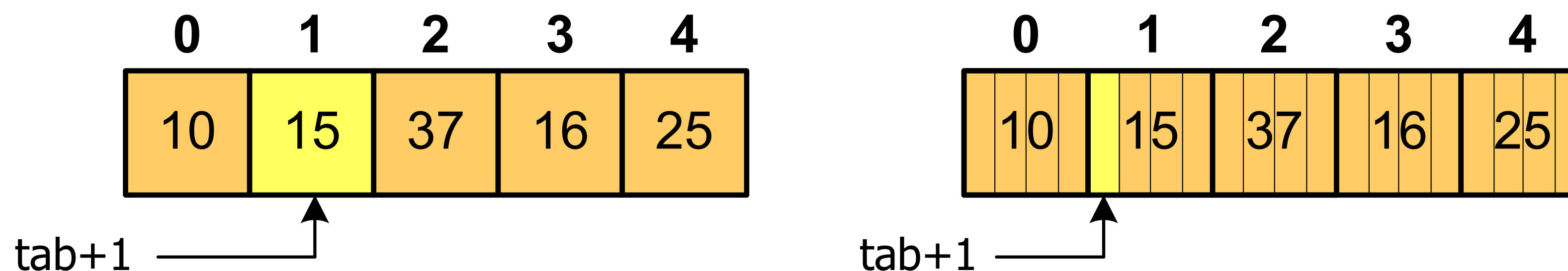


- Zastosowanie operatora * przed nazwą tablicy pozwala „dostać się" do zawartości elementu o indeksie 0

*tab jest równoważne tab[0]

Wskaźniki a tablice

- Dodanie 1 do adresu tablicy przenosi nas do elementu tablicy o indeksie 1 (przesunięcie o 4 bajty, gdyż int zajmuje 4 bajty)



zatem: *(tab+1) jest równoważne tab[1]

ogólnie: *(tab+i) jest równoważne tab[i]

- W zapisie *(tab+i) nawiasy są konieczne, gdyż operator * ma bardzo wysoki priorytet

  x = *tab+1; jest równoważne x = tab[0]+1;

Wskaźniki i struktury

- Gdy zmienna strukturalna jest wskaźnikiem, to do odwołania do pola struktury używamy operatora pośredniego wyboru pola (**->**)

```
wskaźnik_do_struktury -> nazwa_pola
```

```
struct osoba Nowak, *Nowak1;
Nowak1 = &Nowak;
Nowak1 -> wiek = 25;

/* lub */

(*Nowak1).wiek = 25;
```

```
struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
};
```

- W ostatnim zapisie nawiasy są konieczne, gdyż operator **.** ma wyższy priorytet niż operator *****

Dynamiczny przydział pamięci w języku C

- Kiedy stosuje się dynamiczny przydział pamięci?
  - gdy rozmiar tablicy będzie znany dopiero podczas wykonania programu a nie podczas jego kompilacji
  - gdy rozmiar tablicy jest bardzo duży
- Do dynamicznego przydziału pamięci stosowane są funkcje:
  - calloc()
  - malloc()
- Przydział pamięci następuje w obszarze sterty (stosu zmiennych dynamicznych)
- Przydzieloną pamięć należy zwolnić wywołując funkcję:
  - free()

Dynamiczny przydział pamięci w języku C

CALLOC stdlib.h

```
void *calloc(size_t num, size_t size);
```

- Przydziela blok pamięci o rozmiarze num*size (mogący pomieścić tablicę num-elementów, każdy rozmiaru size)
- Zwraca wskaźnik do przydzielonego bloku pamięci
- Jeśli pamięci nie można przydzielić, to zwraca wartość NULL
- Przydzielona pamięć jest inicjowana zerami (bitowo)
- Zwracaną wartość wskaźnika należy rzutować na właściwy typ

```
int *tab;
tab = (int *) calloc(10,sizeof(int));
```

Dynamiczny przydział pamięci w języku C

MALLOC stdlib.h

```
void *malloc(size_t size);
```

- Przydziela blok pamięci o rozmiarze określonym parametrem size
- Zwraca wskaźnik do przydzielonego bloku pamięci
- Jeśli pamięci nie można przydzielić, to zwraca wartość NULL
- Przydzielona pamięć nie jest inicjowana
- Zwracaną wartość wskaźnika należy rzutować na właściwy typ

```
int *tab;
tab = (int *) malloc(10*sizeof(int));
```

Dynamiczny przydział pamięci w języku C

FREE stdlib.h

```
void *free(void *ptr);
```

- Zwalnia blok pamięci wskazywany parametrem ptr
- Wartość ptr musi być wynikiem wywołania funkcji calloc() lub malloc()

```
int *tab;
tab = (int *) calloc(10,sizeof(int));
/* ... */
free(tab);
```

Przykład: przydział pamięci na jedną zmienną

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

int main(void)
{
    float *wsk;

    wsk = (float *) calloc(1,sizeof(float));
    if (wsk == NULL)
    {
        printf("Blad przydzialu pamieci\n");
        return 0;
    }

    *wsk = 123.45f;
    printf("wartosc = %g\n",*wsk);

    free(wsk);
    return 0;
}
```

```
wartosc = 123.45
```

Przykład: przydział pamięci na strukturę

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

struct punkt
{
    int x, y;
};

int main(void)
{
    struct punkt p, *wsk_p;

    wsk_p = (struct punkt*) malloc(sizeof(struct punkt));

    p.x = 10; p.y = 20;
    wsk_p->x = 30; wsk_p->y = 40;
    printf("%d,%d - %d,%d\n",p.x,p.y,wsk_p->x,wsk_p->y);

    free(wsk_p);
    return 0;
}
```

```
10,20 - 30,40
```

Przykład: przydział pamięci na wektor

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

int main(void)
{
    int *tab, n = 10;

    tab = (int *) calloc(n,sizeof(int));

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

    free(tab);

    return 0;
}
```

```
tab[0] = 0
tab[1] = 1
tab[2] = 4
tab[3] = 9
tab[4] = 16
tab[5] = 25
tab[6] = 36
tab[7] = 49
tab[8] = 64
tab[9] = 81
```

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

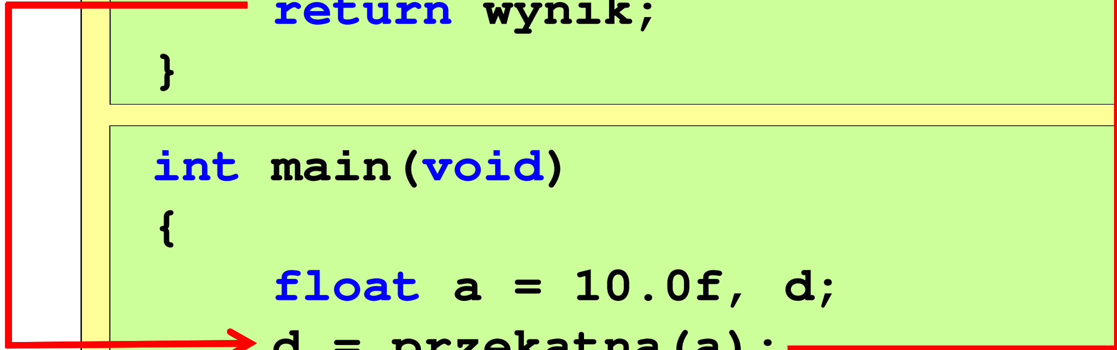
- Program w języku C składa się z funkcji i zmiennych
  - funkcje zawierają instrukcje wykonujące operacje
  - zmienne przechowują wartości

```
#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;
}
```

wywołania funkcji

Funkcje w języku C

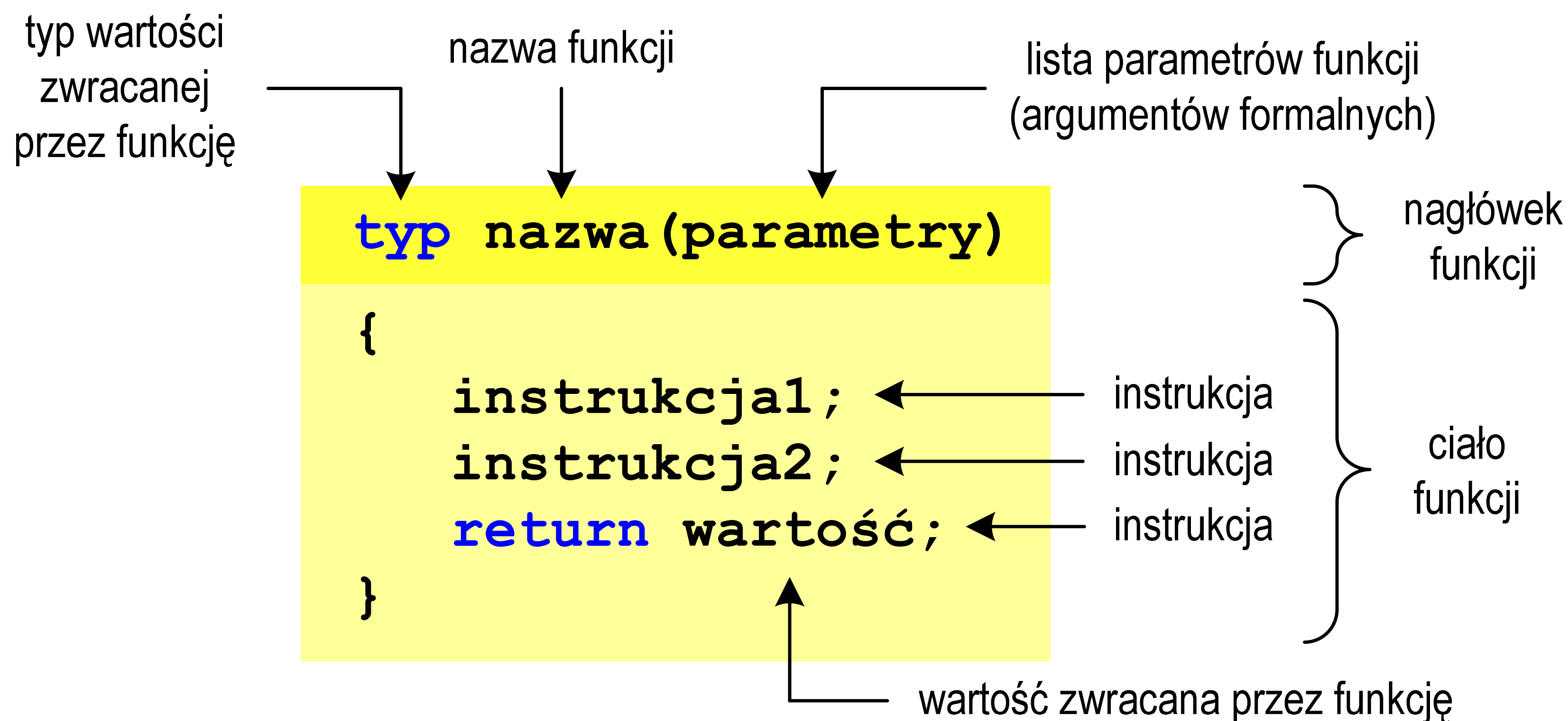
```
#include <stdio.h>     /* przekątna kwadratu */
#include <math.h>

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;
}
```



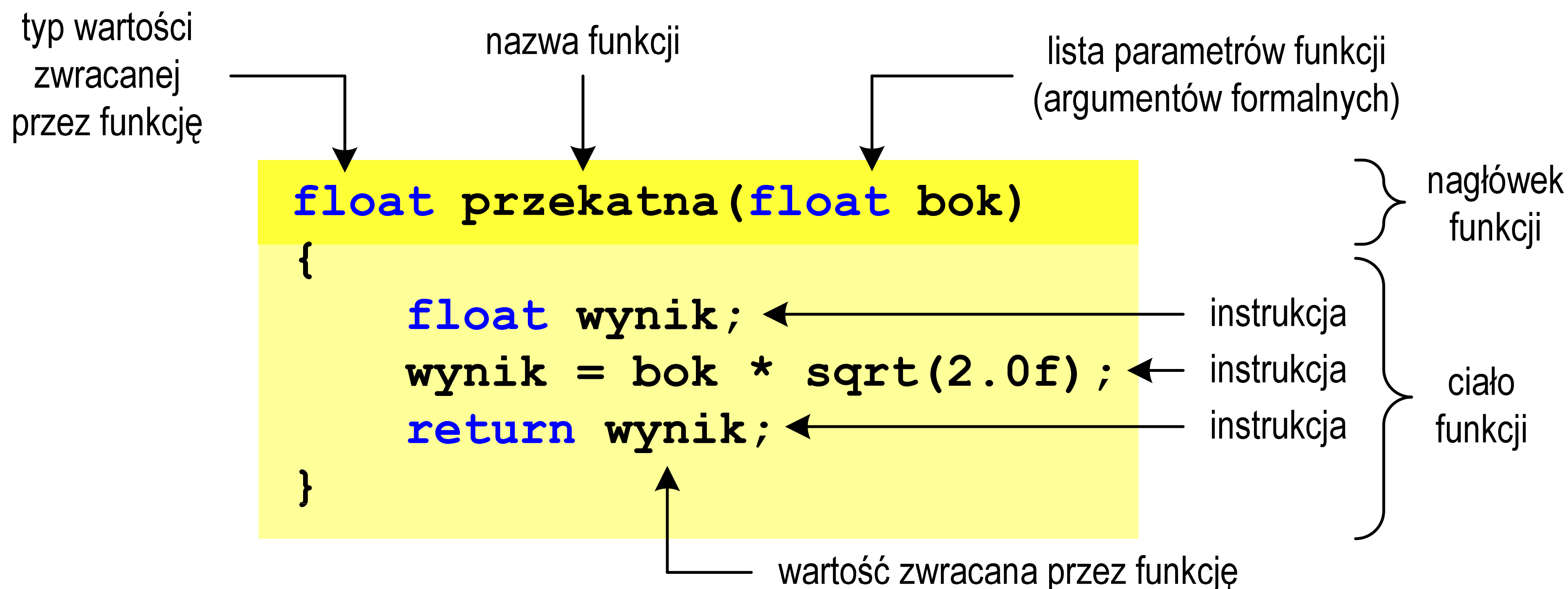
definicja funkcji

definicja funkcji

Ogólna struktura funkcji w języku C



Ogólna struktura funkcji w języku C



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;
}
```

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

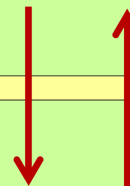
Prototyp funkcji

- Czy można zmienić kolejność definicji funkcji w kodzie programu?

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

definicja funkcji

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



definicja funkcji

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

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

definicja funkcji

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);

int main(void)
{
    float a = 10.0f, b = 5.5f, d;

    d = przekatna(a,b);
    printf("Przekatna prostokata = %g\n",d);

    return 0;
}
```

prototyp funkcji

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",2023-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:    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

```
#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;
}
```

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

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] = {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)
- 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

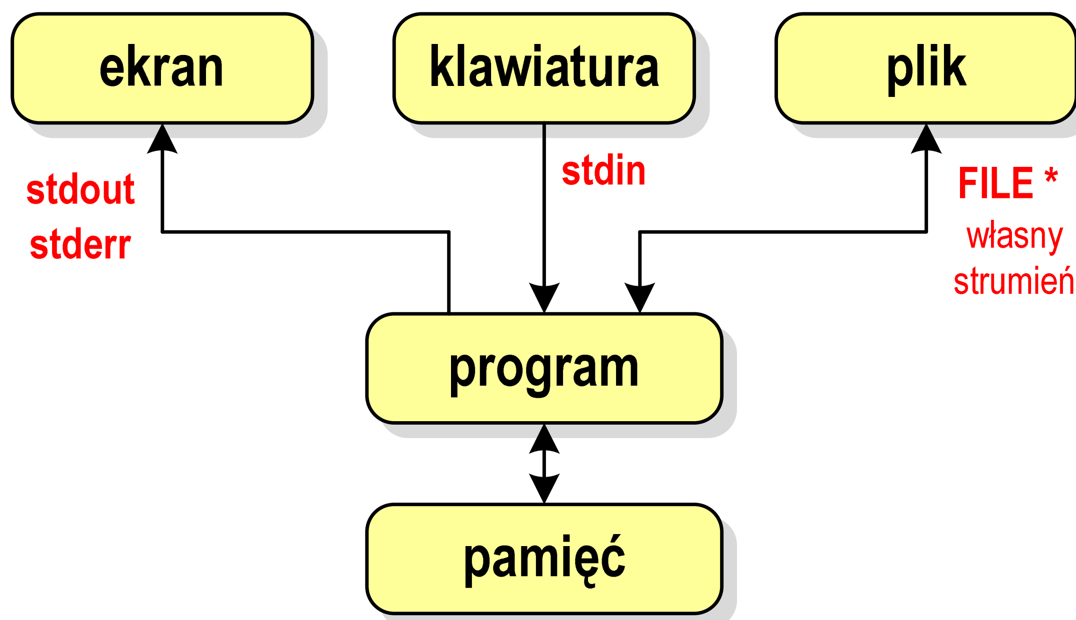
- W każdym programie automatycznie tworzone są i otwierane trzy standardowe strumienie wejścia-wyjścia:
  - stdin - standardowe wejście, skojarzone z klawiaturą
  - stdout - standardowe wyjście , skojarzone z ekranem monitora
  - stderr - standardowe wyjście dla komunikatów o błędach, skojarzone z ekranem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

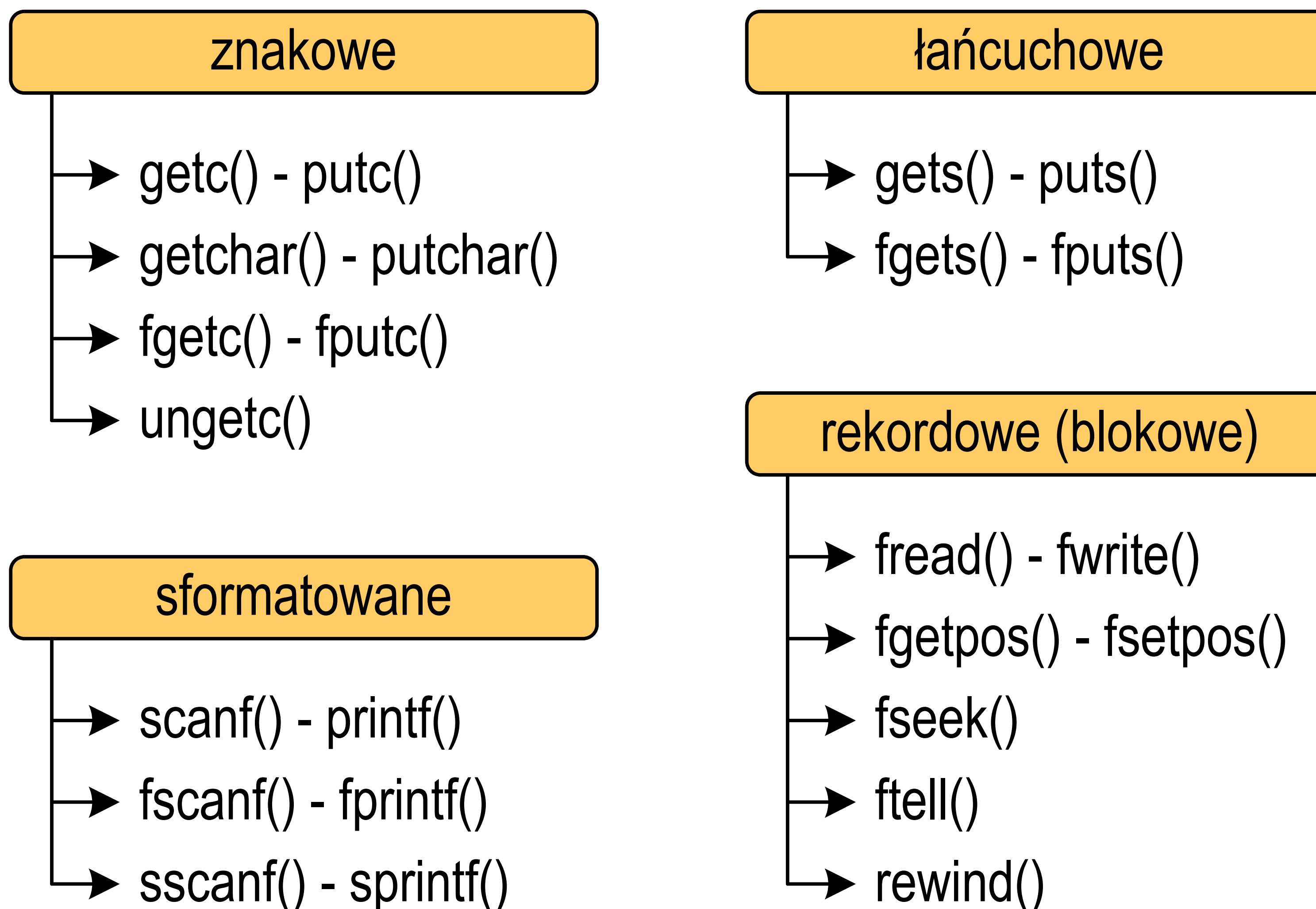
Strumienie

- Współpraca programu z „otoczeniem”

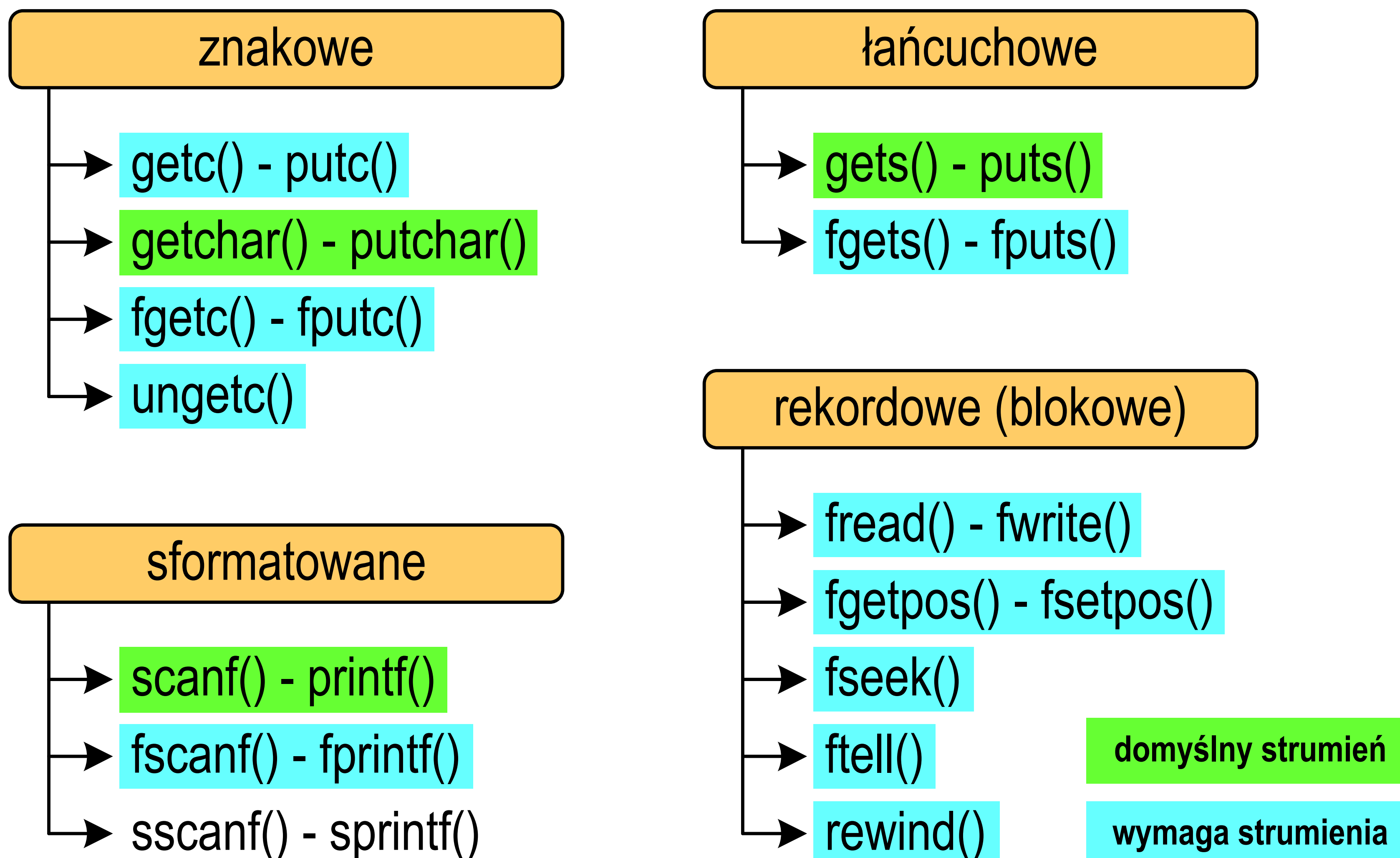


- Standardowe funkcje wejścia-wyjścia mogą:
    - domyślnie korzystać z określonego strumienia (stdin, stdout, stderr)
    - wymagać podania strumienia (własnego, stdin, stdout, stderr)

Typy standardowych operacji wejścia-wyjścia



Typy standardowych operacji wejścia-wyjścia



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
```

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 stdin)

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

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
```

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
```

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
```

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("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):

```
MZ.  └   ┘   ··   ¸       @                                  č   ♬.ş♬ ´.Í❢¸.
LÍ❢This program cannot be run in DOS mode....$      {9◘ő?Xf¦?Xf¦?Xf¦❢.ő¦<X
f¦❢.Í¦,Xf¦↑ž.¦=Xf¦?Xg¦└Xf¦❢.â¦7Xf¦❢.ň¦>Xf¦❢.÷¦>Xf¦Rich?Xf¦
       PE  L.● .^ZR       ŕ ┐.♂..  8   :      ↑◄.  ┼  ┼   @ ┼  ┐  │
    │        °.  ┘      └ @.  ┼  ┼   ┼  ┼    ┼           €. <    .. Ś│
                    . ◘└  W. .                                          °.
 . t.                            .textbss  . ┼                      ŕ.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■
```

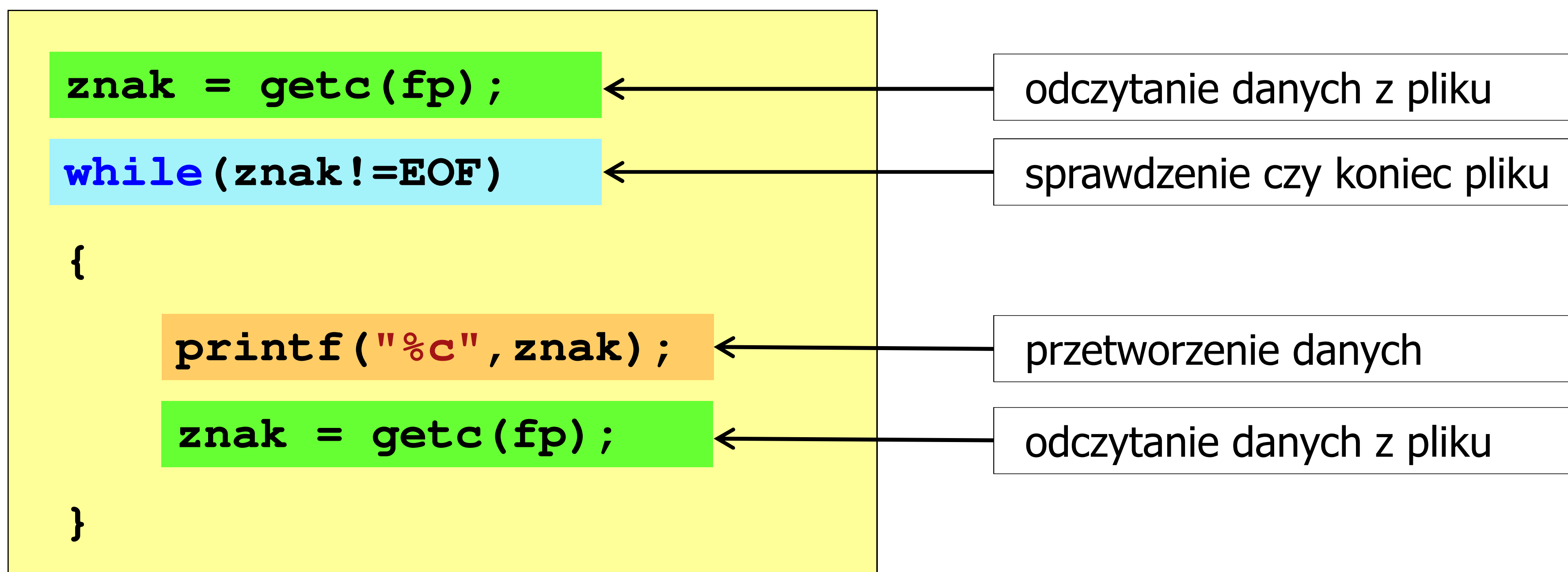
- Pliki binarne nie mają ściśle określonej struktury

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



Koniec wykładu nr 8

Dziękuję za uwagę!