

Programowanie C (CP1S01005)

Politechnika Białostocka - Wydział Elektryczny
Cyfryzacja przemysłu, sem. I, studia stacjonarne I stopnia
Rok akademicki 2023/2024

Wykład nr 3 (10.11.2023)

dr inż. Jarosław Forenc

Plan wykładu nr 3

- Pętle while i do...while
- Funkcje w języku C
 - ogólna struktura funkcji
 - argumenty i parametry funkcji
 - domyślne wartości parametrów funkcji
 - prototypy funkcji
 - typy funkcji
 - przekazywanie argumentów do funkcji przez wartość i przez wskaźnik
 - funkcje rekurencyjne

Przykład: pierwiastek kwadratowy

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

int main(void)
{
    float x, y;

    printf("Podaj liczbe: ");
    scanf("%f", &x);

    if (x>=0)
    {
        y = sqrt(x);
        printf("Pierwiastek liczby: %f\n", y);
    }
    else
        printf("Blad! Liczba ujemna\n");

    return 0;
}
```

Podaj liczbe: -3
Blad! Liczba ujemna

Podaj liczbe: 3
Pierwiastek liczby: 1.732051

Przykład: pierwiastek kwadratowy (pętla while)

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

int main(void)
{
    float x, y;

    printf("Podaj liczbe: ");
    scanf("%f", &x);
    while (x<0)
    {
        printf("Blad! Liczba ujemna\n\n");
        printf("Podaj liczbe: ");
        scanf("%f", &x);
    }
    y = sqrt(x);
    printf("Pierwiastek liczby: %f\n", y);

    return 0;
}
```

```
Podaj liczbe: -3
Blad! Liczba ujemna
```

```
Podaj liczbe: -5
Blad! Liczba ujemna
```

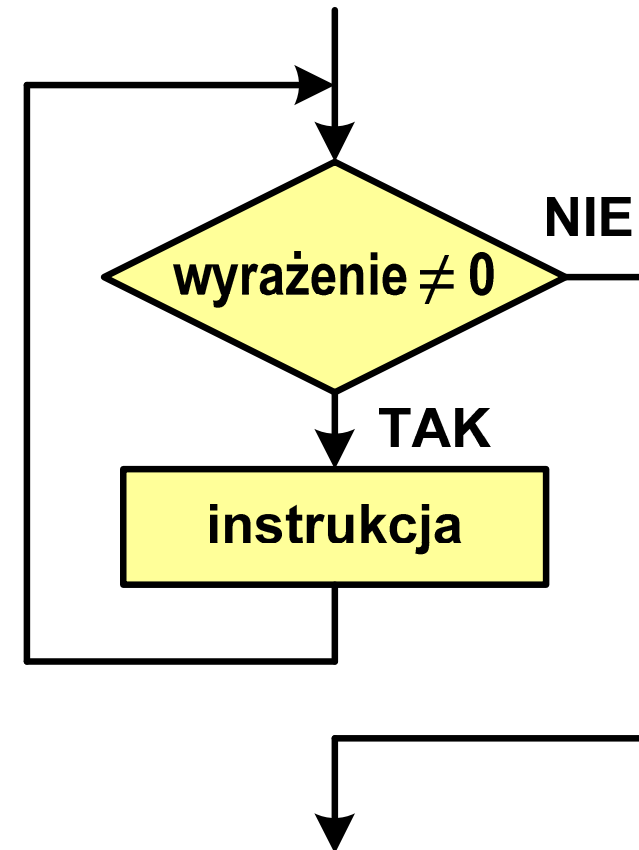
```
Podaj liczbe: 3
Pierwiastek liczby: 1.732051
```

Język C - pętla while

```
while (wyrażenie)  
    instrukcja;
```

- „dopóki wyrażenie w nawiasach jest prawdziwe wykonuj instrukcję”

- Wyrażenie w nawiasach:
 - **prawdziwe** - gdy jego wartość jest różna od zera
 - **fałszywe** - gdy jego wartość jest równa zero
- Jako wyrażenie najczęściej stosowane jest **wyrażenie logiczne**



Język C - pętla while

```
while (wyrażenie)  
    instrukcja;
```

- Instrukcja:
 - **prosta** - jedna instrukcja zakończona średnikiem
 - **złożona** - jedna lub kilka instrukcji objętych nawiasami klamrowymi

```
int x = 10;  
while (x>0)  
    x = x - 1;
```

```
int x = 10;  
while (x>0)  
{  
    printf("%d\n", x);  
    x = x - 1;  
}
```

Przykład: suma liczb dodatnich

```
#include <stdio.h>

int main(void)
{
    int x, suma = 0;

    printf("Podaj liczbe: ");
    scanf("%d", &x);

    while (x>0)
    {
        suma = suma + x;
        printf("Podaj liczbe: ");
        scanf("%d", &x);
    }
    printf("Suma liczb: %d\n", suma);

    return 0;
}
```

```
Podaj liczbe: 4
Podaj liczbe: 8
Podaj liczbe: 2
Podaj liczbe: 3
Podaj liczbe: 5
Podaj liczbe: -2
Suma liczb: 22
```

Język C - pętla while

- Program pokazany na poprzednim slajdzie zawiera typowy schemat przetwarzania danych z wykorzystaniem pętli **while**

```
printf("Podaj liczbę: ");  
scanf("%d", &x);
```

wczytanie danych

```
while (x>0)
```

```
{
```

```
    suma = suma + x;
```

operacje na danych

```
    printf("Podaj liczbę: ");  
    scanf("%d", &x);
```

wczytanie danych

```
}
```

- Dane mogą być wczytywane z klawiatury, pliku, itp.

Język C - pętla while (break, continue)

- **break** i **continue** są to instrukcje skoku

```
int x=0;
while (x<10)
{
    x++;
    if (x%2==0)
        continue;
    if (x%5==0)
        break;
    printf ("%d\n", x);
}
```

- **continue** przerywa bieżącą iterację
- **break** przerywa wykonywanie pętli

Język C - pętla while (najczęstsze błędy)

- Postawienie średnika po wyrażeniu w nawiasach powoduje powstanie pętli nieskończonej - program zatrzymuje się na pętli

```
int x = 10;  
while (x>0);  
    printf("%d ", x--);
```



- Brak aktualizacji zmiennej powoduje także powstanie pętli nieskończonej - program wyświetla wielokrotnie tę samą wartość

```
int x = 10;  
while (x>0)  
    printf("%d ", x);
```

10 10 10 10 10 ...

Język C - pętla while (pętla nieskończona)

- W pewnych sytuacjach celowo stosuje się pętlę nieskończoną (np. w mikrokontrolerach)

```
while (1)
{
    instrukcja;
    instrukcja;
    ...
}
```

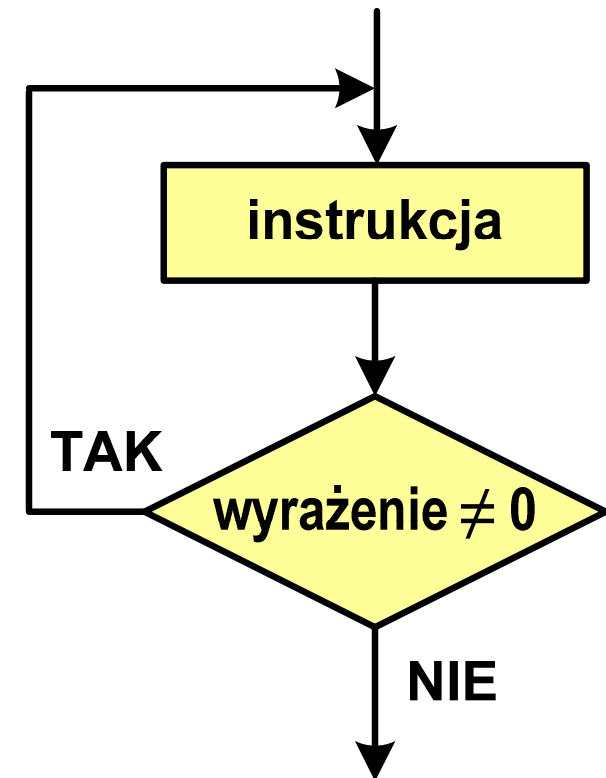
- W układach mikroprocesorowych program działa aż do wyłączenia zasilania

Język C - pętla do ... while

```
do  
    instrukcja;  
while (wyrażenie);
```

- „wykonuj instrukcję dopóki wyrażenie w nawiasach jest prawdziwe”

- Wyrażenie w nawiasach:
 - **prawdziwe** - gdy jego wartość jest różna od zera
 - **fałszywe** - gdy jego wartość jest równa zero



Język C - pętla do ... while

```
do
    instrukcja;
while (wyrażenie);
```

- Instrukcja:
 - **prosta** - jedna instrukcja zakończona średnikiem
 - **złożona** - jedna lub kilka instrukcji objętych nawiasami klamrowymi

```
int x = 10;
do
    x = x - 1;
while (x>0);
```

```
int x = 10;
do
{
    printf("%d\n", x);
    x = x - 1;
}
while (x>0);
```

Język C - pętla do ... while (break, continue)

- **break** i **continue** są to instrukcje skoku

```
int x=0;

do
{
    x++;
    if (x%5==0)
        break;
    if (x%2==0)
        continue;
    printf ("%d\n", x);
}
while (i<10);
```

- **break** przerywa wykonywanie pętli
- **continue** przerywa bieżącą iterację

Przykład: suma liczb < 100

```
#include <stdio.h>

int main(void)
{
    int x, suma = 0;

    do
    {
        printf("Podaj liczbe: ");
        scanf("%d", &x);
        suma = suma + x;
    }
    while (suma < 100);

    printf("Suma liczb: %d\n", suma);

    return 0;
}
```

```
Podaj liczbe: 34
Podaj liczbe: 9
Podaj liczbe: 26
Podaj liczbe: -8
Podaj liczbe: 67
Suma liczb: 128
```

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>      /* przekatna 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>      /* przekatna 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

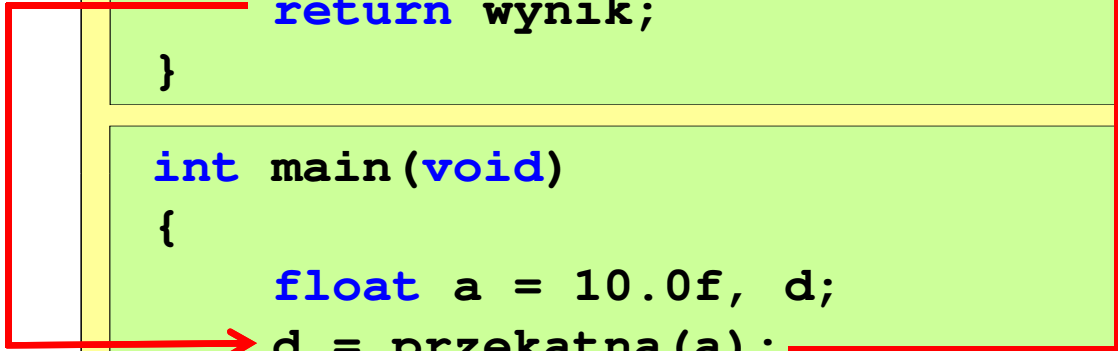
```
#include <stdio.h>      /* przekatna kwadratu */  
#include <math.h>
```

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

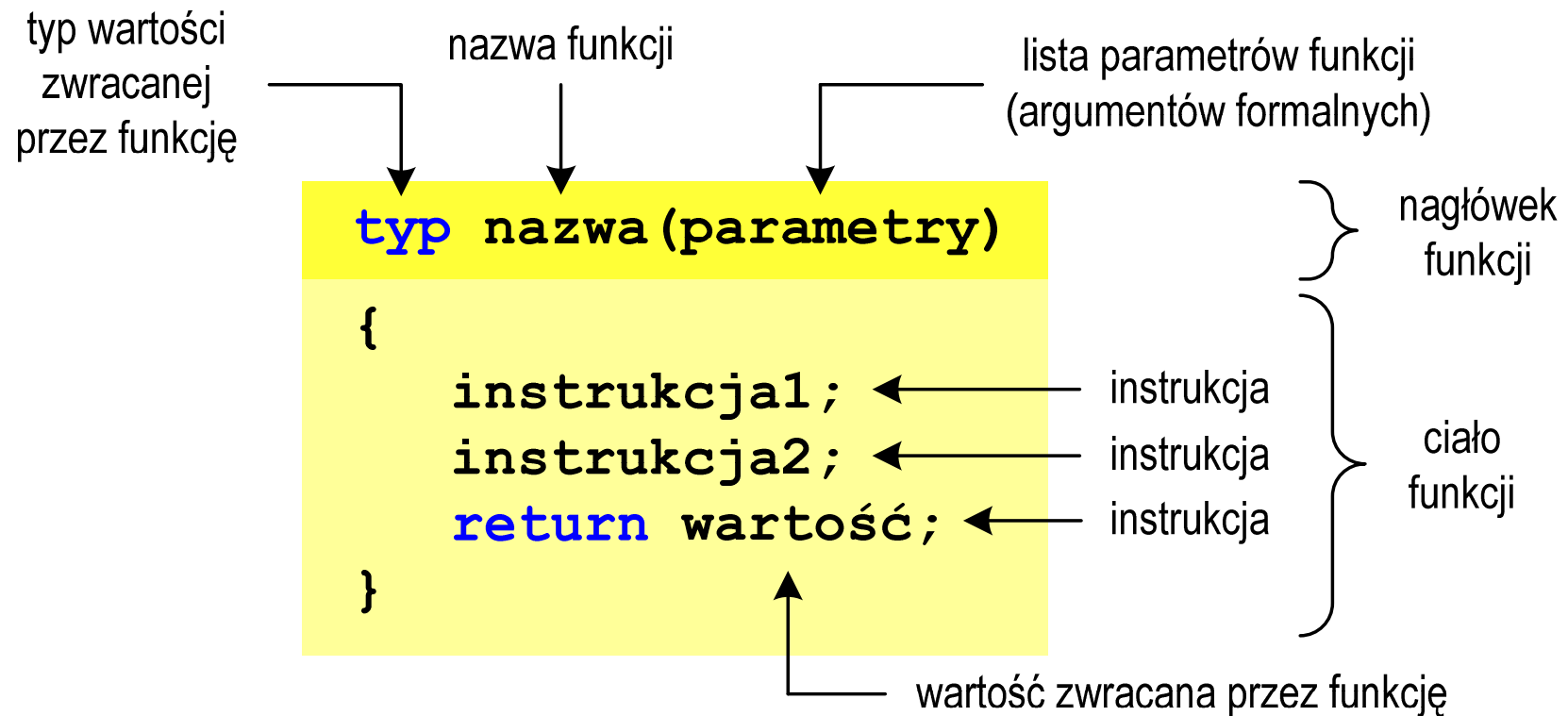
definicja funkcji

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

definicja funkcji



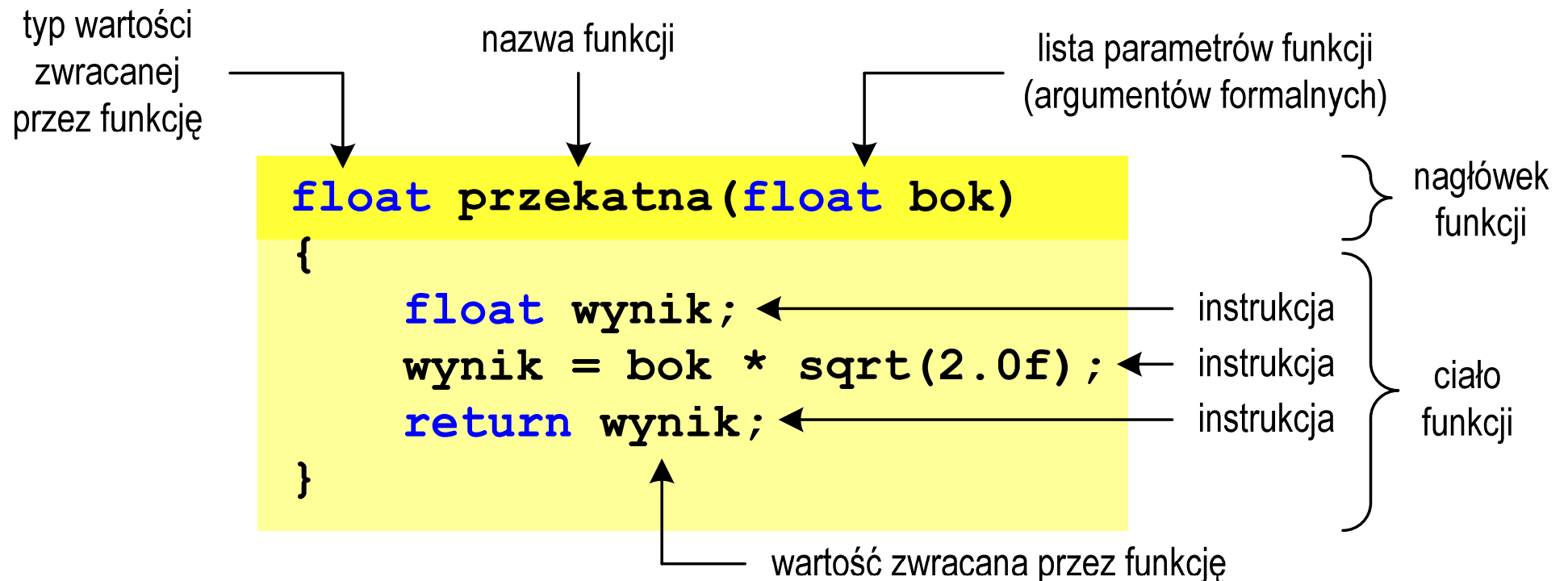
Ogólna struktura funkcji w języku C



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

lista argumentów funkcji
(argumentów faktycznych)

Ogólna struktura funkcji w języku C



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

lista argumentów funkcji
(argumentów faktycznych)

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>      /* przekatna prostokata */
#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

71-80 pkt. → 4,0

51-60 pkt. → 3,0

81-90 pkt. → 4,5

61-70 pkt. → 3,5

0-50 pkt. → 2,0

Prototyp funkcji

- Czy można zmienić kolejność definicji funkcji w kodzie programu?

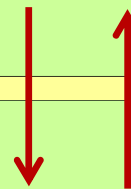
```
#include <stdio.h>      /* przekątna prostokąta */
#include <math.h>
```

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

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;
}
```

definicja funkcji



Prototyp funkcji

- Czy można zmienić kolejność definicji funkcji w kodzie programu?

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

definicja funkcji

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

definicja funkcji

Prototyp funkcji

- Czy można zmienić kolejność definicji funkcji w kodzie programu?

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

definicja funkcji

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

error C3861: 'przekatna':
identifier not found

Prototyp funkcji

```
#include <stdio.h>      /* przekatna prostokata */  
#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", 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

```
#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 przez wartość

```
#include <stdio.h>

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

int main(void)
{
    int a = 20;

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

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	20	main()

Przekazywanie argumentów przez wartość

```
#include <stdio.h>

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

int main(void)
{
    int a = 20;

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

    return 0;
}
```

Fragment pamięci komputera

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

Przekazywanie argumentów przez wartość

```
#include <stdio.h>

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

int main(void)
{
    int a = 20;

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

    return 0;
}
```

Fragment pamięci komputera

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

fun: a = 10

Przekazywanie argumentów przez wartość

```
#include <stdio.h>

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

int main(void)
{
    int a = 20;

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

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	20	main()

```
fun: a = 10
main: a = 20
```

Przekazywanie argumentów przez wskaźnik

```
#include <stdio.h>

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

int main(void)
{
    int a = 20;

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

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	20	main()

Przekazywanie argumentów przez wskaźnik

```
#include <stdio.h>

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

int main(void)
{
    int a = 20;

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

    return 0;
}
```

Fragment pamięci komputera

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

Przekazywanie argumentów przez wskaźnik

```
#include <stdio.h>

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

int main(void)
{
    int a = 20;

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

    return 0;
}
```

Fragment pamięci komputera

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

fun: a = 10

Przekazywanie argumentów przez wskaźnik

```
#include <stdio.h>

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

int main(void)
{
    int a = 20;

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

    return 0;
}
```

Fragment pamięci komputera

	Adres zmiennej	Wartość	
a	0x0024FBDC	10	main()

```
fun: a = 10
main: a = 10
```

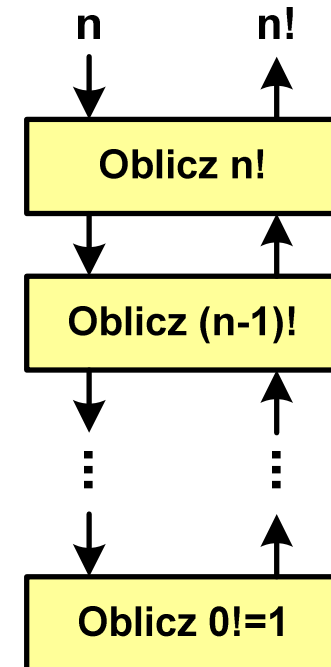
Funkcje rekurencyjne

- **Rekurencja** lub **rekursja** - jest to odwoływanie się funkcji lub definicji do samej siebie
- Rozwiązanie danego problemu wyraża się za pomocą rozwiązań tego samego problemu, ale dla danych o mniejszych rozmiarach
- W matematyce mechanizm rekurencji stosowany jest do definiowania lub opisywania algorytmów

- Silnia:

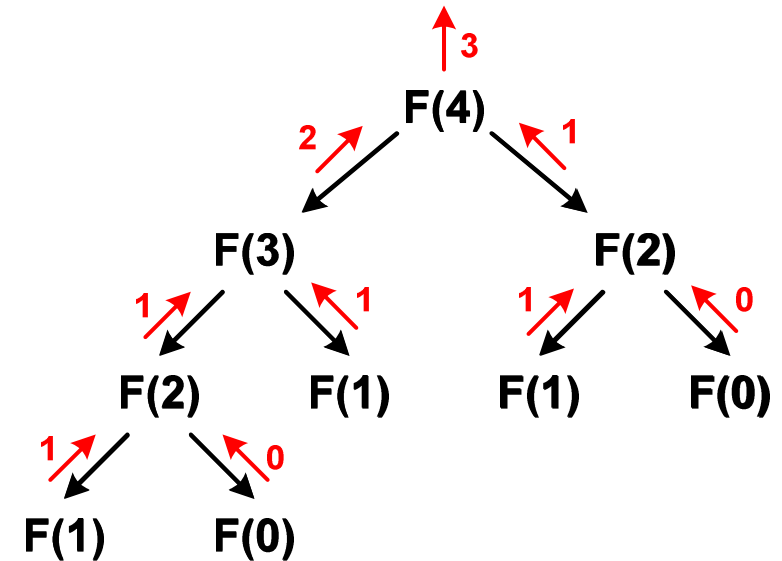
$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ n(n-1)! & \text{dla } n \geq 1 \end{cases}$$

```
int silnia(int n)
{
    return n==0 ? 1 : n*silnia(n-1);
}
```



Rekurencja - ciąg Fibonacciego

$$F_n = \begin{cases} 0 & \text{dla } n = 0 \\ 1 & \text{dla } n = 1 \\ F_{n-1} + F_{n-2} & \text{dla } n > 1 \end{cases}$$



```
int F(int n)
{
    if (n==0) return 0;
    if (n==1) return 1;
    return F(n-1) + F(n-2);
}
```

Koniec wykładu nr 3

Dziękuję za uwagę!