

Informatyka 2 (ES1D300 017)

Politechnika Białostocka - Wydział Elektryczny
Elektrotechnika, semestr III, studia stacjonarne I stopnia
Rok akademicki 2019/2020

Wykład nr 5 (29.10.2019)

dr inż. Jarosław Forenc

Plan wykładu nr 5

- Funkcje w języku C
 - ogólna struktura funkcji
 - argumenty i parametry funkcji
 - domyślne wartości parametrów funkcji
 - wskaźniki do funkcji
 - wywołanie funkcji przez wskaźnik
- Prototypy funkcji, typy 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>    /* 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> /* 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;
}
```

wywołania funkcji

Funkcje w języku C

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

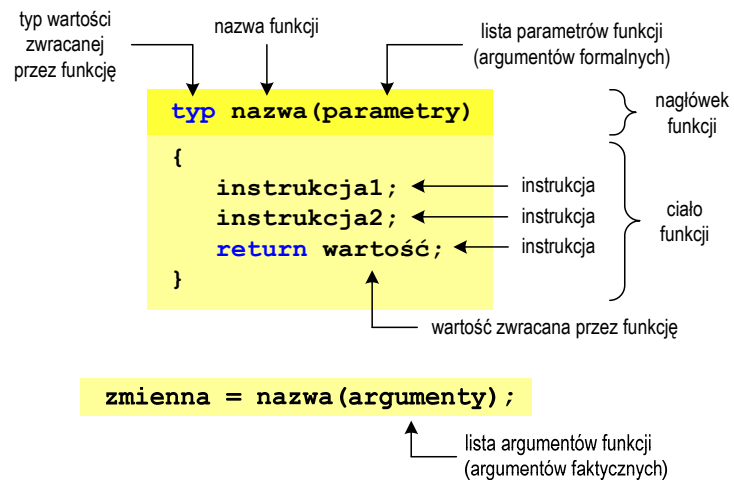
float przekatna(float bok)
{
    float wynik;
    wynik = bok * sqrt(2.0f);
    return wynik;
}

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

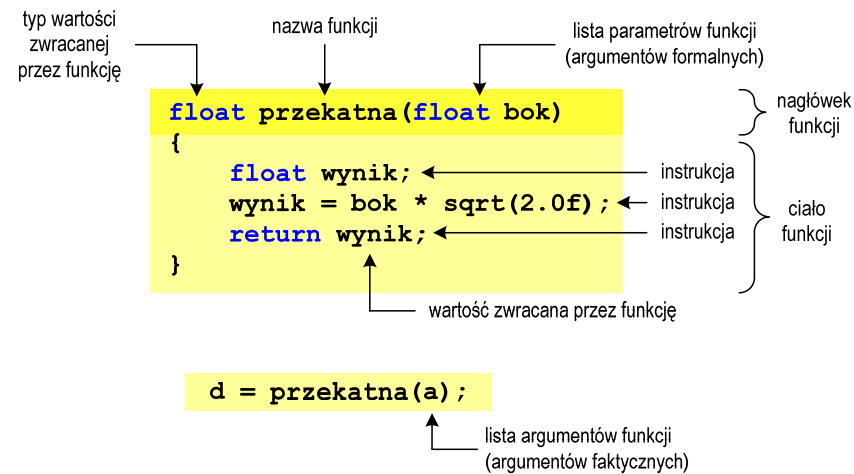
definicja funkcji

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

```
/* przekatna 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 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

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

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> /* 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;
}
```

definicja funkcji

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;
}

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> /* 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;
}

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", 2019-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:           %s\n", imie);
    printf("Nazwisko:        %s\n", nazwisko);
    printf("Wiek:             %d\n", wiek);
    printf("Rok urodzenia:    %d\n\n", 2019-wiek);
}

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

    return 0;
}
```

```
Imie:           Jan
Nazwisko:        Kowalski
Wiek:           23
Rok urodzenia:  1996

Imie:           Barbara
Nazwisko:        Nowak
Wiek:           28
Rok urodzenia:  1991
```

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

Koniec wykładu nr 5

Dziękuję za uwagę!