

Funkcje

czyli jak programować proceduralne.

Ostatnia aktualizacja: 17 września 2025

STRUKTURA PROGRAMU W C

```
#include <stdio.h>
#define PI 3.1415
```

Dyrektywy preprocesora

```
float g = 2.5;
```

Zmienne globalne

```
float kwadrat(float x)
{
    return x*x;
}
```

Definicja funkcji

```
int main()
{
```

Funkcja główna

```
    float x, y;
```

Zmienne lokalne

```
    x = PI * g;
    y = kwadrat(x);
```

Instrukcje programu

```
    return 0;
```

```
}
```

PODPROGRAM

wdzielony fragment programu (kodu) zawierający instrukcje do wielokrotnego użytku

- Dekompozycja złożonego problemu na prostsze
- Przejrzystość
- Unikanie powtórzeń kodu
- Uniwersalność - jak najmniejszy związek z konkretnym kodem
- Biblioteki funkcji - zbiory funkcji ogólnego zastosowania
- Rekurencja(Rekurencja(Rekurencja(Rekurencja(...))))
- Nic za darmo - wywołanie funkcji to dodatkowy koszt czasu i pamięci

DEKLARACJA FUNKCJI (PROTOTYP)

Zapowiedź użycia funkcji - określenie nazwy funkcji, typów argumentów i typu wartości zwracanej

```
typ identyfikator(typ arg1, typ arg2, ... );
```

DEFINICJA FUNKCJI

Deklaracja parametrów i instrukcje podprogramu (ciało funkcji).

```
typ identyfikator(typ arg1, typ arg2, ... )  
{  
    deklaracje zmiennych lokalnych  
    instrukcje  
    return wartość;  
}
```

DEKLARACJE

```
float sin(float x);  
  
float pierwiastek(float x);  
  
void wypiszmenu(void);  
  
int random(void);  
  
int nwd(int a, int b);  
  
char getchar(void);  
  
int fff(int z, char z, float a);
```

WYWOŁANIE

```
y = sin(x);  
  
y = pierwiastek(5.0);  
  
wypiszmenu();  
  
i = random();  
  
i = nwd(144, a + 1);  
  
z = getchar();  
  
i = fff(3, 'A', 3.14);
```

PRZYKŁAD FUNKCJI NWD

```
1  #include <stdio.h>
2
3  int nwd(int a, int b)
4  {
5      int c;
6      while (b != 0)
7      {
8          c = a % b;
9          a = b;
10         b = c;
11     }
12     return a;
13 }
14
15 int main()
16 {
17     int a, b;
18
19     printf("Podaj dwie liczby calkowite: ");
20     scanf("%d %d", &a, &b);
21     printf("NWD(%d,%d) = %d\n", a, b, nwd(a,b));
22
23     return 0;
24 }
```

PARAMETRY FORMALNE I AKTUALNE FUNKCJI

PARAMETRY FORMALNE

```
int nwd(int a, int b)
{
    int c;
    ...
}
```

PARAMETRY AKTUALNE

```
float x,y;
int a=1, b=2, c;

x = sin(1);
c = nwd(144, b);
y = sin(x * nwd(1+a, nwd(100, b))));
```

W języku C argumenty są przekazywane wyłącznie przez wartość, tzn. wartość argumentu jest kopiowana do parametru formalnego.

FUNKCJA BEZ WARTOŚCI ZWRACANEJ - PROCEDURA

```
void wypisz(int n)
{
    while( n>0 )
    {
        printf("Programowaie proceduralne\n");
        n = n - 1;
    }
}
```

FUNKCJA Z WARTOŚCIĄ ZWRACANĄ

```
int silnia(int n)
{
    int i=2;  x=1;
    while( i <= n )
    {
        x = x * i;
        i = i + 1;
    }
    return x;
}
```


- return przerywa działanie funkcji i zwraca wartość z funkcji
- Może pojawić się w dowolnym miejscu wewnątrz funkcji

```
int jest_pierwsza(int n)
{
    int i=2;
    while( i<n )
    {
        if( n % i == 0 ) return 0;
        i = i + 1;
    }
    return 1;
}
```

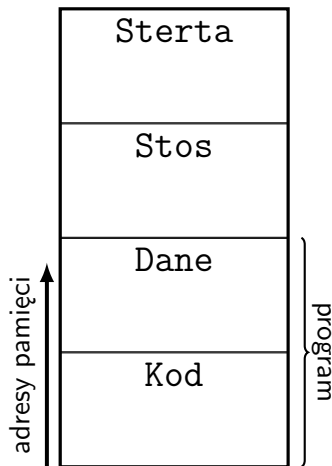
Wartość zwracana z funkcji jest podstawiona w miejsce wywołania

```
if(jest_pierwsza(a)) printf("%d jest pierwsza\n",a);
```

DEKLARACJA I DEFINICJA FUNKCJI C.D

```
1 #include <stdio.h>
2
3 /* Deklaracja funkcji */
4 int jest_pierwsza ( int n );
5
6 int main()
7 {
8     int n, i=1;
9     printf("n = ");
10    scanf("%d", &n);
11    while(i<=n)
12    {
13        if(jest_pierwsza(i) == 1) printf("%d\n",i);
14        i++;
15    }
16    return 0;
17 }
18
19 /* Definicja funkcji */
20 int jest_pierwsza ( int n )
21 {
22     int i =2;
23     while ( i<=n/i )
24     {
25         if ( n % i == 0 ) return 0 ;
26         i = i + 1;
27     }
28     return 1 ;
29 }
```

W momencie kompilacji musi być znana przynajmniej deklaracja funkcji (jej nazwa i typy argumentów).



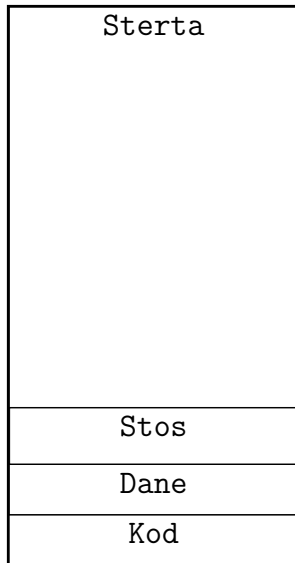
pamięć dostępna do alokacji

zmienne lokalne funkcji wkładane
na stos przy wywołaniu funkcji

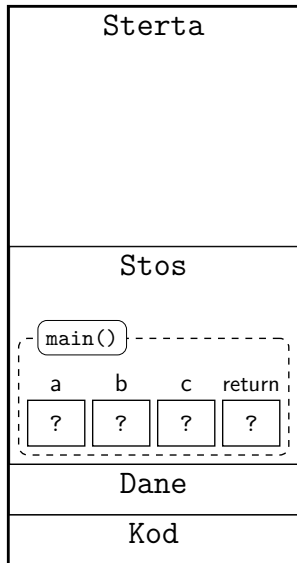
stałe oraz **zmienne globalne** i
statyczne inicjowane przy
uruchomieniu

tekst programu
tylko do odczytu

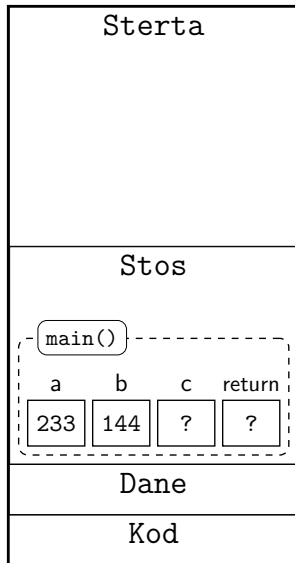
```
1  int nwd(int a, int b)
2  {
3      int c;
4      while (b != 0)
5      {
6          c = a % b;
7          a = b;
8          b = c;
9      }
10     return a;
11 }
12
13 int main()
14 {
15     int a, b, c;
16     a = 233;
17     b = 144;
18     c = nwd(a, b);
19     return 0;
20 }
```



```
1  int nwd(int a, int b)
2  {
3      int c;
4      while (b != 0)
5      {
6          c = a % b;
7          a = b;
8          b = c;
9      }
10     return a;
11 }
12
13 int main()
14 {
15     int a, b, c;
16     a = 233;
17     b = 144;
18     c = nwd(a, b);
19     return 0;
20 }
```



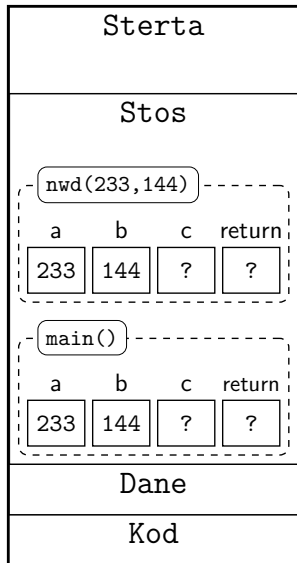
```
1  int nwd(int a, int b)
2  {
3      int c;
4      while (b != 0)
5      {
6          c = a % b;
7          a = b;
8          b = c;
9      }
10     return a;
11 }
12
13 int main()
14 {
15     int a, b, c;
16     a = 233;
17     b = 144;
18     c = nwd(a, b);
19     return 0;
20 }
```



```

1  int nwd(int a, int b)
2  {
3      int c;
4      while (b != 0)
5      {
6          c = a % b;
7          a = b;
8          b = c;
9      }
10     return a;
11 }
12
13 int main()
14 {
15     int a, b, c;
16     a = 233;
17     b = 144;
18     c = nwd(a, b);
19     return 0;
20 }

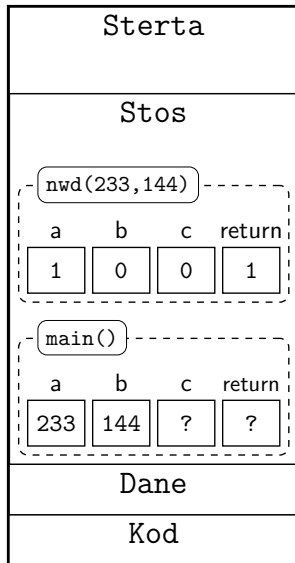
```



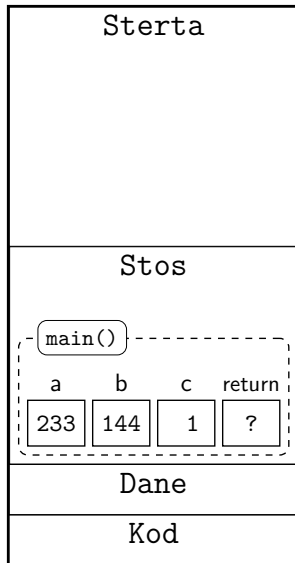
```

1  int nwd(int a, int b)
2  {
3      int c;
4      while (b != 0)
5      {
6          c = a % b;
7          a = b;
8          b = c;
9      }
10     return a;
11 }
12
13 int main()
14 {
15     int a, b, c;
16     a = 233;
17     b = 144;
18     c = nwd(a, b);
19     return 0;
20 }

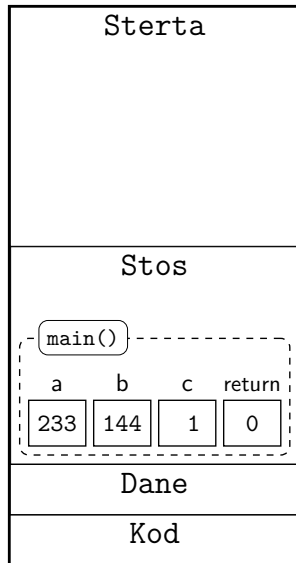
```



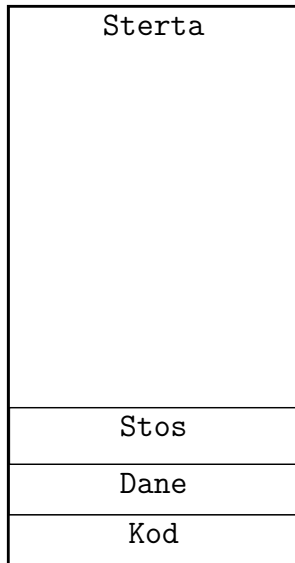

```
1  int nwd(int a, int b)
2  {
3      int c;
4      while (b != 0)
5      {
6          c = a % b;
7          a = b;
8          b = c;
9      }
10     return a;
11 }
12
13 int main()
14 {
15     int a, b, c;
16     a = 233;
17     b = 144;
18     c = nwd(a, b);
19     return 0;
20 }
```



```
1  int nwd(int a, int b)
2  {
3      int c;
4      while (b != 0)
5      {
6          c = a % b;
7          a = b;
8          b = c;
9      }
10     return a;
11 }
12
13 int main()
14 {
15     int a, b, c;
16     a = 233;
17     b = 144;
18     c = nwd(a, b);
19     return 0;
20 }
```



```
1  int nwd(int a, int b)
2  {
3      int c;
4      while (b != 0)
5      {
6          c = a % b;
7          a = b;
8          b = c;
9      }
10     return a;
11 }
12
13 int main()
14 {
15     int a, b, c;
16     a = 233;
17     b = 144;
18     c = nwd(a, b);
19     return 0;
20 }
```



ZASIĘG ZMIENNYCH: GLOBALNE I LOKALNE

ZMIENNA GLOBALNA

dostępna dla wszystkich funkcji programu. Zmienna istnieje przez cały czas działania programu. Deklaracja zmiennej nie znajduje się wewnątrz żadnej funkcji.

```
int a;  
void funkcja()  
{  
    a = a + 1;  
}
```

ZMIENNA LOKALNA (AUTOMATYCZNA)

wewnętrzna zmienna funkcji. Czas życia ograniczony czasem działania funkcji.

```
void funkcja(int a)  
{  
    a = a + 1;  
}
```

```
1  #include<stdio.h>
2
3  int globalna = 0;
4
5  void f(void)
6  {
7      int lokalna = 0;
8      globalna = globalna + 1;
9      lokalna = lokalna + 1;
10     printf("%d\t %d\t f\n", globalna, lokalna);
11 }
12
13 int main()
14 {
15     int lokalna = 13;
16
17     printf("globalna lokalna funkcja\n");
18     printf("%d\t %d\t main\n", globalna, lokalna);
19     f();
20     printf("%d\t %d\t main\n", globalna, lokalna);
21     f();
22     printf("%d\t %d\t main\n", globalna, lokalna);
23     return 0;
24 }
```

 globalne.c

```

1  #include<stdio.h>
2
3  int globalna = 0;
4
5  void f(void)
6  {
7      int lokalna = 0;
8      globalna = globalna + 1;
9      lokalna = lokalna + 1;
10     printf("%d\t %d\t f\n", globalna, lokalna);
11 }
12
13 int main()
14 {
15     int lokalna = 13;
16
17     printf("globalna lokalna funkcja\n");
18     printf("%d\t %d\t main\n", globalna, lokalna);
19     f();
20     printf("%d\t %d\t main\n", globalna, lokalna);
21     f();
22     printf("%d\t %d\t main\n", globalna, lokalna);
23     return 0;
24 }

```

Wynik działania:

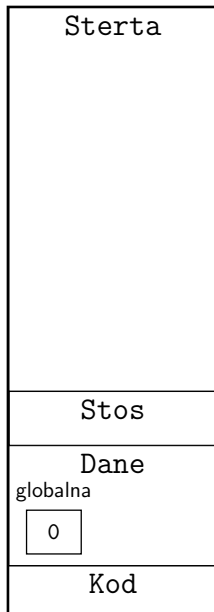
globalna	lokalna	funkcja
0	13	main
1	1	f
1	13	main
2	1	f
2	13	main

 globalne.c

```

1  #include<stdio.h>
2
3  int globalna = 0;
4
5  void f(void)
6  {
7      int lokalna = 0;
8      globalna = globalna + 1;
9      lokalna = lokalna + 1;
10 }
11
12 int main()
13 {
14     int lokalna = 13;
15     globalna = globalna + 1;
16     f();
17
18     return 0;
19 }

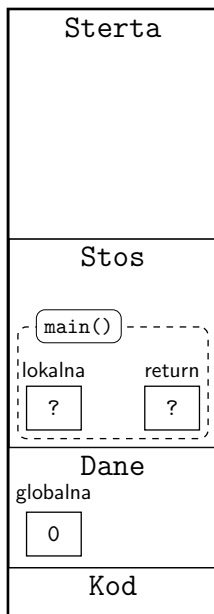
```



```

1  #include<stdio.h>
2
3  int globalna = 0;
4
5  void f(void)
6  {
7      int lokalna = 0;
8      globalna = globalna + 1;
9      lokalna = lokalna + 1;
10 }
11
12 int main()
13 {
14     int lokalna = 13;
15     globalna = globalna + 1;
16     f();
17
18     return 0;
19 }

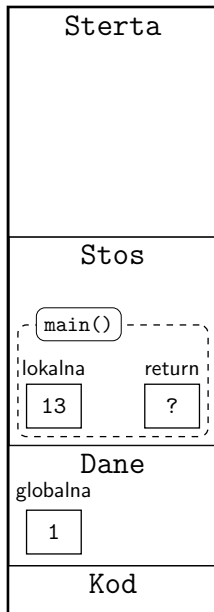
```




```

1  #include<stdio.h>
2
3  int globalna = 0;
4
5  void f(void)
6  {
7      int lokalna = 0;
8      globalna = globalna + 1;
9      lokalna = lokalna + 1;
10 }
11
12 int main()
13 {
14     int lokalna = 13;
15     globalna = globalna + 1;
16     f();
17
18     return 0;
19 }

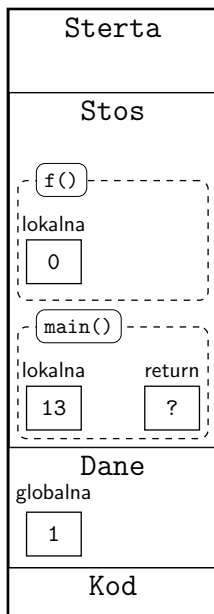
```



```

1  #include<stdio.h>
2
3  int globalna = 0;
4
5  void f(void)
6  {
7      int lokalna = 0;
8      globalna = globalna + 1;
9      lokalna = lokalna + 1;
10 }
11
12 int main()
13 {
14     int lokalna = 13;
15     globalna = globalna + 1;
16     f();
17
18     return 0;
19 }

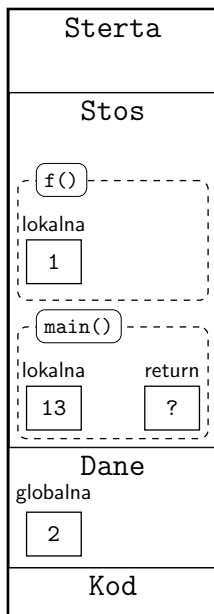
```



```

1  #include<stdio.h>
2
3  int globalna = 0;
4
5  void f(void)
6  {
7      int lokalna = 0;
8      globalna = globalna + 1;
9      lokalna = lokalna + 1;
10 }
11
12 int main()
13 {
14     int lokalna = 13;
15     globalna = globalna + 1;
16     f();
17
18     return 0;
19 }

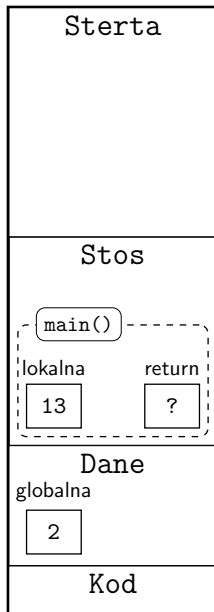
```

```

1  #include<stdio.h>
2
3  int globalna = 0;
4
5  void f(void)
6  {
7      int lokalna = 0;
8      globalna = globalna + 1;
9      lokalna = lokalna + 1;
10 }
11
12 int main()
13 {
14     int lokalna = 13;
15     globalna = globalna + 1;
16     f();
17
18     return 0;
19 }

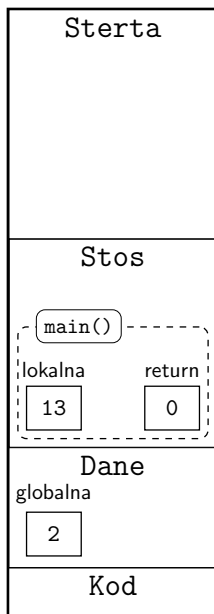
```



```

1  #include<stdio.h>
2
3  int globalna = 0;
4
5  void f(void)
6  {
7      int lokalna = 0;
8      globalna = globalna + 1;
9      lokalna = lokalna + 1;
10 }
11
12 int main()
13 {
14     int lokalna = 13;
15     globalna = globalna + 1;
16     f();
17
18     return 0;
19 }

```



W programowaniu proceduralnym nie używamy zmiennych globalnych!

```
#include<stdio.h>
int x=1, y=1;

int main()
{
    f();
    zzz();
    pewna_funkcja();

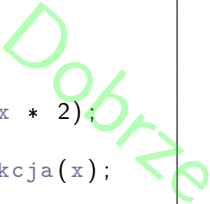
    printf("%d %d\n", x, y);
}
```



```
#include<stdio.h>

int main()
{
    int x=1, y=1;
    y = f(x, 10, x * 2);
    zzz();
    x = pewna_funkcja(x);

    printf("%d %d\n", x, y);
}
```



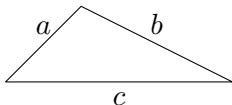
<code><assert.h></code>	Asercje - diagnostyka kodu
<code><ctype.h></code>	Klasyfikacja znaków
<code><float.h></code>	Ograniczenia typów zmiennopozycyjnych
<code><limits.h></code>	Ograniczenia typów całkowitych
<code><signal.h></code>	Obsługa sygnałów
<code><stddef.h></code>	Standardowe definicje (makra)
<code><stdio.h></code>	Operacje wejścia/wyjścia
<code><math.h></code>	Funkcje matematyczne
<code><stdlib.h></code>	Zestaw podstawowych narzędzi, np. <code>rand()</code>
<code><string.h></code>	Obsługa łańcuchów znakowych
<code><time.h></code>	Funkcje obsługi czasu

sin	cos	tan	funkcje trygonometryczne
asin	acos	atan	
sinh	cosh	tanh	funkcje hiperboliczne
exp			f. eksponencjalna e^x
ceil	floor		zaokrąglenia: <i>sufit</i> , <i>podłoga</i>
sqrt			pierwiastek $\sqrt{x}$
pow			potęga x^y
log			logarytm naturalny $\ln(x) = \log_e x$
log10			logarytm dziesiętny $\log_{10} x$
fabs			wartość bezwzględna $ x $
fmod			reszta z dzielenia (zmiennoprzecinkowe)
HUGE_VAL			bardzo duża wartość double

Uwaga: korzystając z kompilatora GCC należy dodać opcję `-lm`
`gcc -lm euclid.c`

PRZYKŁAD: WZÓR HERONA

Problem: pole trójkąta dla danych długości boków a , b i c .



WZÓR HERONA (60 N.E.)

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

gdzie

$$p = \frac{1}{2}(a+b+c)$$

W jakiej sytuacji wyrażenie pod pierwiastkiem jest mniejsze od 0 ?

PRZYKŁAD: POLE TRÓJKĄTA

```
1  #include <math.h>
2
3  /*  Funkcja wyznacza pole trojkata z wzoru Heroana.
4      *  Argumenty a, b i c to dlugosci bokow.
5      *  Jezeli boki a, b, i c nie tworza trojkata
6      *  zwracana jest wartosc -1. */
7  float heron(float a, float b, float c)
8  {
9      float p = (a+b+c)/2;
10     p = p*(p-a)*(p-b)*(p-c);
11     if(p<0) return -1;
12     return sqrt(p);
13 }
```

 heron2.c

```

1  int main()
2  {
3      float a, b, c, pole;
4
5      printf("Podaj dlugosci bokow trojkata: a, b, c > 0\n");
6      scanf("%f%f%f", &a, &b, &c);
7
8      if ( a <= 0 || b <= 0 || c<=0 )
9      {
10         printf("Zle dane: wartosci musza byc dodatnie.\n");
11         return 1;
12     }
13
14     pole = heron(a, b, c);
15     if( pole < 0 )
16     {
17         printf("Zle dane: to nie sa boki trojkata\n");
18         return 2;
19     }
20     printf("Pole wynosi: %f\n", pole);
21
22     return 0;
23 }

```

 heron2.c

Funkcje mogą wywoływać same siebie - **funkcje rekurencyjne**.

```
#include<stdio.h>

void funkcja()
{
    /* ciało funkcji */
    funkcja();
}

int main()
{
    funkcja();
}
```

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n = n \cdot (n - 1)!$$

```

1  #include<stdio.h>
2
3  int silnia(int n)
4  {
5      if( n<=1 ) return 1;
6      return n * silnia(n-1);
7  }
8
9  int main()
10 {
11     int x, n=3;
12     x = silnia(n);
13     printf ("%d!=%d\n",n,x);
14     return 0;
15 }

```

Sterna
Stos
Dane
Kod

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n = n \cdot (n - 1)!$$

```

1  #include<stdio.h>
2
3  int silnia(int n)
4  {
5      if( n<=1 ) return 1;
6      return n * silnia(n-1);
7  }
8
9  int main()
10 {
11     int x, n=3;
12     x = silnia(n);
13     printf ("%d!=%d\n", n, x);
14     return 0;
15 }

```

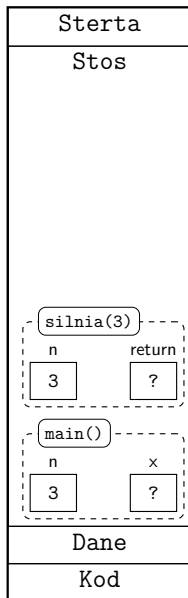


$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n = n \cdot (n - 1)!$$

```

1  #include<stdio.h>
2
3  int silnia(int n)
4  {
5      if( n<=1 ) return 1;
6      return n * silnia(n-1);
7  }
8
9  int main()
10 {
11     int x, n=3;
12     x = silnia(n);
13     printf ("%d!=%d\n",n,x);
14     return 0;
15 }

```

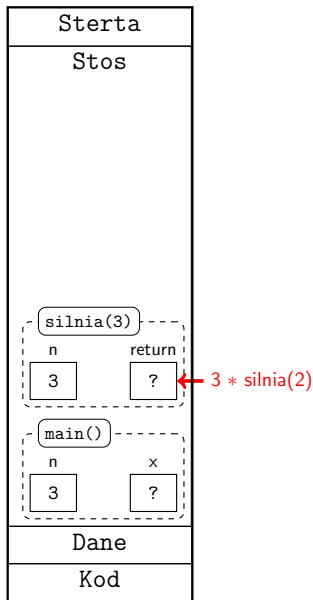


$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n = n \cdot (n - 1)!$$

```

1  #include<stdio.h>
2
3  int silnia(int n)
4  {
5      if( n<=1 ) return 1;
6      return n * silnia(n-1);
7  }
8
9  int main()
10 {
11     int x, n=3;
12     x = silnia(n);
13     printf ("%d!=%d\n", n, x);
14     return 0;
15 }

```

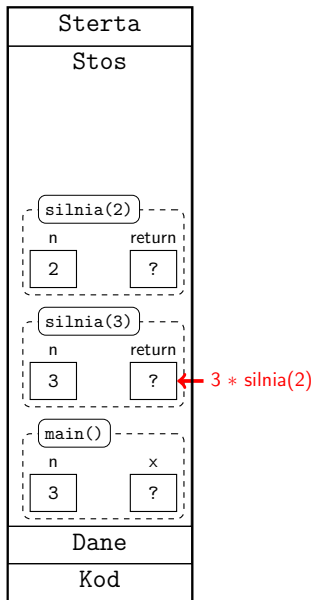


$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n = n \cdot (n - 1)!$$

```

1  #include<stdio.h>
2
3  int silnia(int n)
4  {
5      if( n<=1 ) return 1;
6      return n * silnia(n-1);
7  }
8
9  int main()
10 {
11     int x, n=3;
12     x = silnia(n);
13     printf ("%d!=%d\n", n, x);
14     return 0;
15 }

```

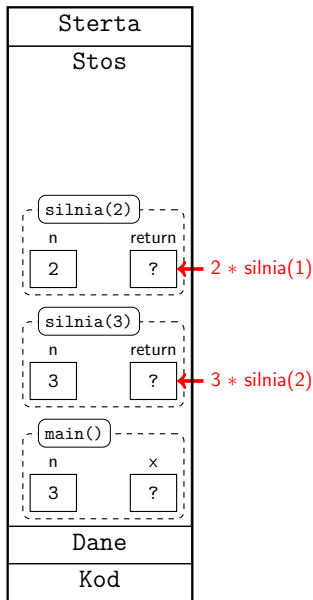


$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n = n \cdot (n - 1)!$$

```

1  #include<stdio.h>
2
3  int silnia(int n)
4  {
5      if( n<=1 ) return 1;
6      return n * silnia(n-1);
7  }
8
9  int main()
10 {
11     int x, n=3;
12     x = silnia(n);
13     printf ("%d!=%d\n", n, x);
14     return 0;
15 }

```

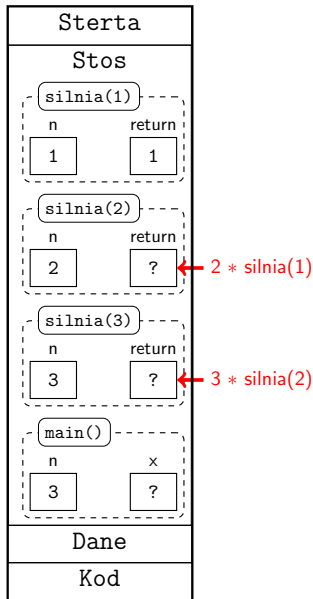


$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n = n \cdot (n - 1)!$$

```

1  #include<stdio.h>
2
3  int silnia(int n)
4  {
5      if( n<=1 ) return 1;
6      return n * silnia(n-1);
7  }
8
9  int main()
10 {
11     int x, n=3;
12     x = silnia(n);
13     printf ("%d!=%d\n", n, x);
14     return 0;
15 }

```

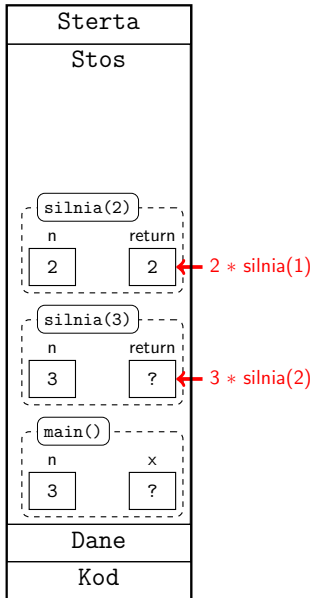


$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n = n \cdot (n - 1)!$$

```

1  #include<stdio.h>
2
3  int silnia(int n)
4  {
5      if( n<=1 ) return 1;
6      return n * silnia(n-1);
7  }
8
9  int main()
10 {
11     int x, n=3;
12     x = silnia(n);
13     printf ("%d!=%d\n", n, x);
14     return 0;
15 }

```

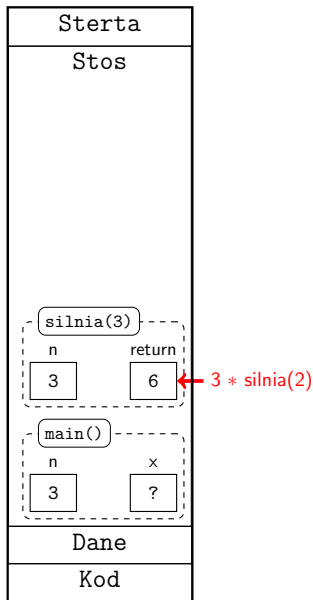


$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n = n \cdot (n - 1)!$$

```

1  #include<stdio.h>
2
3  int silnia(int n)
4  {
5      if( n<=1 ) return 1;
6      return n * silnia(n-1);
7  }
8
9  int main()
10 {
11     int x, n=3;
12     x = silnia(n);
13     printf ("%d!=%d\n", n, x);
14     return 0;
15 }

```

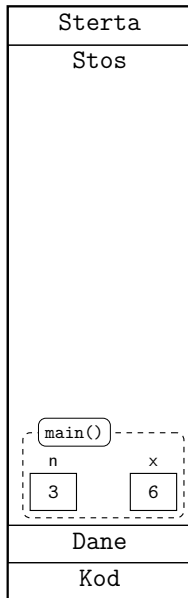


$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n = n \cdot (n - 1)!$$

```

1  #include<stdio.h>
2
3  int silnia(int n)
4  {
5      if( n<=1 ) return 1;
6      return n * silnia(n-1);
7  }
8
9  int main()
10 {
11     int x, n=3;
12     x = silnia(n);
13     printf ("%d!=%d\n", n, x);
14     return 0;
15 }

```



ZADANIE O ROZMNAŻANIU KRÓLIKÓW

Problem: zadanie o rozmnażaniu się królików.

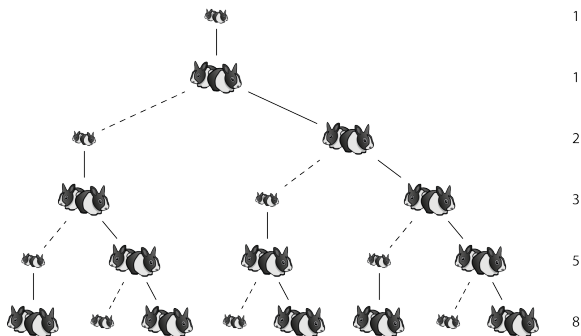
Populacja królików rozmnaża się wg. poniższych zasad:

- rozpoczynamy od pojedynczej pary nowonarodzonych królików,
- króliki stają się płodne po upływie miesiąca życia,
- każda płodna para wydaje na świat parę królików co miesiąc,
- króliki nigdy nie umierają.

Ile par królików będzie w populacji po roku?

ZADANIE O ROZMNAŻANIU KRÓLIKÓW

Ile par królików będzie w populacji po n miesiącach?



Źródło: Jens Bossaert, Curiosa Mathematica,
<http://curiosamathematica.tumblr.com/>

PRZYKŁAD: CIĄG FIBINACCIEGO

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...,

CIĄG FIBINACCIEGO

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

REKURENCJA

```
1 int fibonacci(int n)
2 {
3     if( n==0 ) return 1;
4     if( n==1 ) return 1;
5     return fibonacci(n-1) + fibonacci(n-2);
6 }
```

 fib1.c

ITERACJA

```
1 int fibonacci(int n)
2 {
3     int f=1, fp=1, k;
4
5     while( n>1 )
6     {
7         k = f + fp;
8         fp = f;
9         f = k;
10        n--;
11    }
12    return f;
13 }
```

 fib2.c

- **Zmienne globalne** mają zasięg całego pliku, **zmienne lokalne** istnieją tylko wewnątrz funkcji
- **Programowanie proceduralne:** funkcje nie powinny korzystać ze zmiennych globalnych
- **Deklaracja funkcji** określa nazwę funkcji, typy argumentów i typ wartości zwracanej
- **Definicja funkcji** to deklaracja + ciało funkcji (implementacja)
- Funkcja przed użyciem musi być przynajmniej zadeklarowana
- `return` zwraca pojedynczą wartość z funkcji
- Pytanie: jak zwrócić z funkcji więcej wartości?
Np. funkcja wyznaczająca miejsca zerowe paraboli.

-  David Griffiths, Dawn Griffiths „*Rusz głową! C.*”, Helion, Gliwice, 2013.
-  „Kurs programowania w C”, WikiBooks,
<http://pl.wikibooks.org/wiki/C>
-  „C Reference”, <http://en.cppreference.com/w/c>