Funkcje

czyli jak programować proceduralne.

Ostatnia aktualizacja: 29 września 2026

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

float g = 2.5;

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

int main()
{
    float x, y;

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

    return 0;
}
```

Dyrektywy preprocesora

Zmienne globalne

Definicja funkcji

Funkcja główna

Zmienne lokalne

Instrukcje programu

Podprogram

wydzielony 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;
}
```

Przykłady deklaracji

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 funkcji są przekazywane **wyłącznie przez wartość**, tzn. parametr aktualny jest kopiowany do parametru formalnego. Funkcja opera się na kopii, a nie na oryginalnej zmiennej.

Funkcja bez wartości zwracanej (procedura)

```
void wypisz(int n)
{
    while(n > 0)
    {
            printf("Programowanie 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;
}
```

Instrukcja `return` przerywa działanie funkcji i zwraca wartość z funkcji. Może wystąpić 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

```c
#include <stdio.h>

/* Deklaracja funkcji */
int jest_pierwsza ( int n );

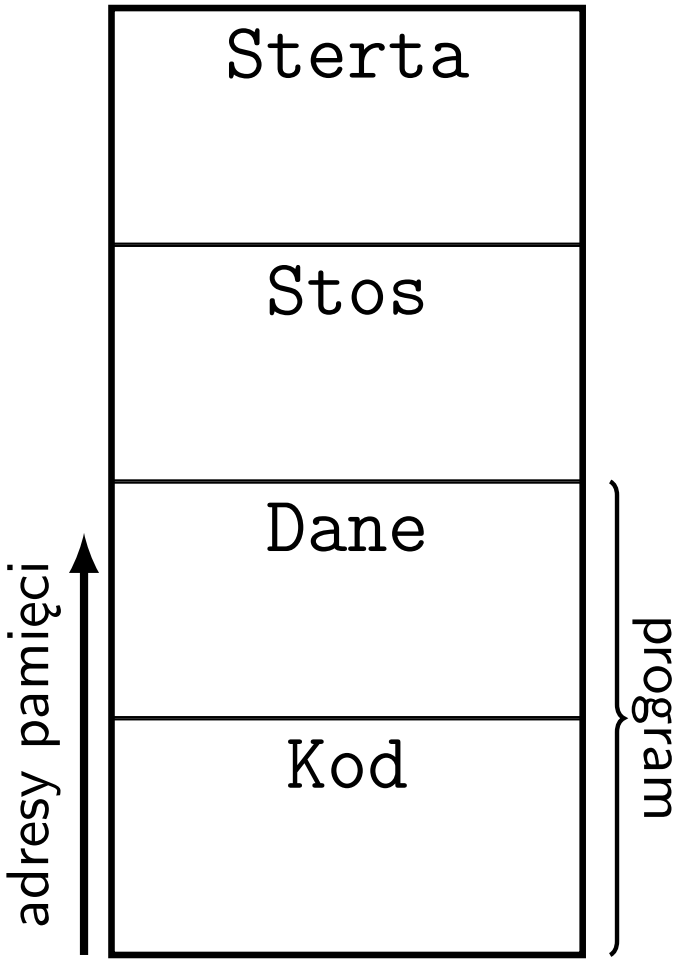
int main()
{
   int n, i=1;
   printf("n = ");
   scanf("%d", &n);
   while(i <= n)
   {
      if(jest_pierwsza(i) == 1) printf("%d\n", i);
      i++;
   }
   return 0;
}

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

☞ pierwsza2.c

W momencie kompilacji pliku muszą być znana przynajmniej deklaracje wszystkich funkcji (nazwy i typy argumentów).

Definicje funkcji mogą wystąpić później, zwykle umieszczane są w osobnym pliku źródłowym.



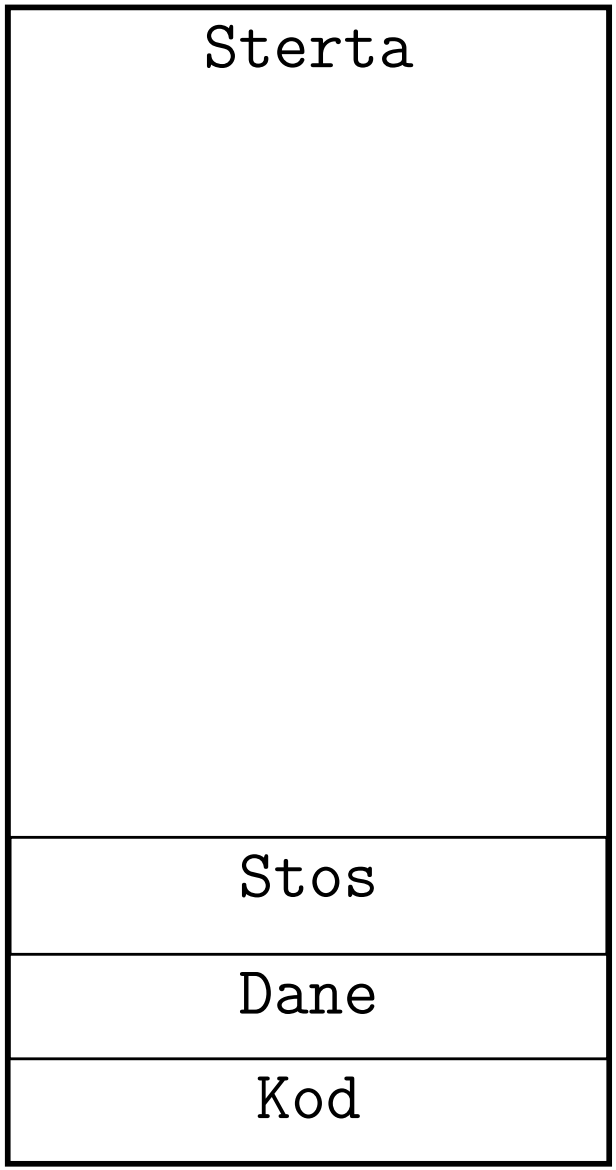
pamięć dostępna do alokacji

zmienne lokalne funkcji wkładane na stos przy wywołaniu funkcji

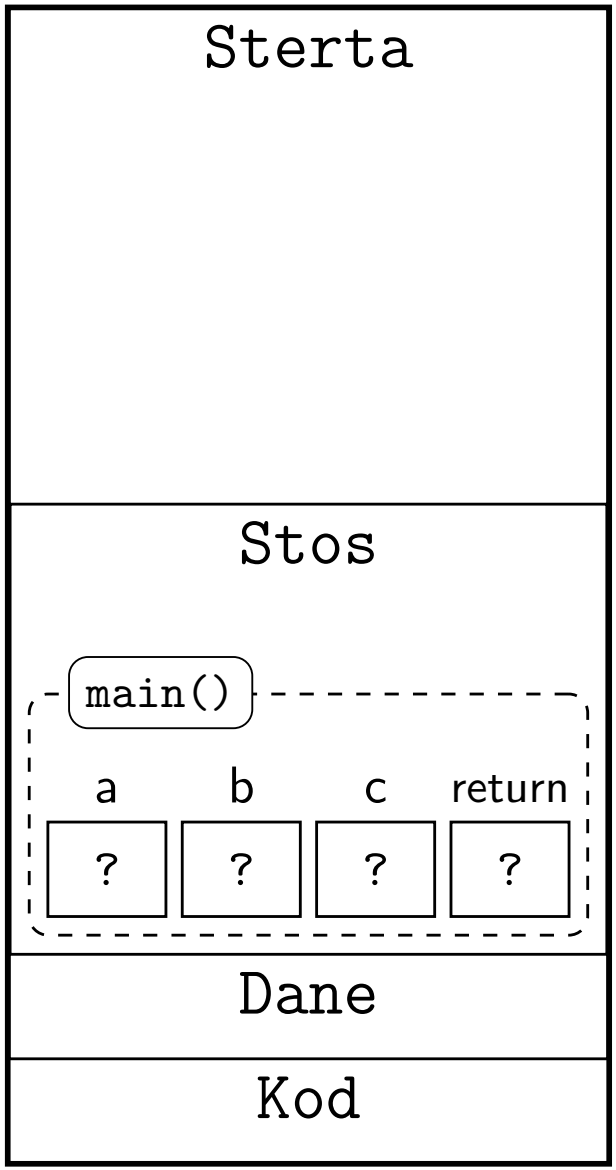
stałe oraz **zmienne globalne** i statyczne inicjowane przy uruchomieniu

tekst programu
tylko do odczytu

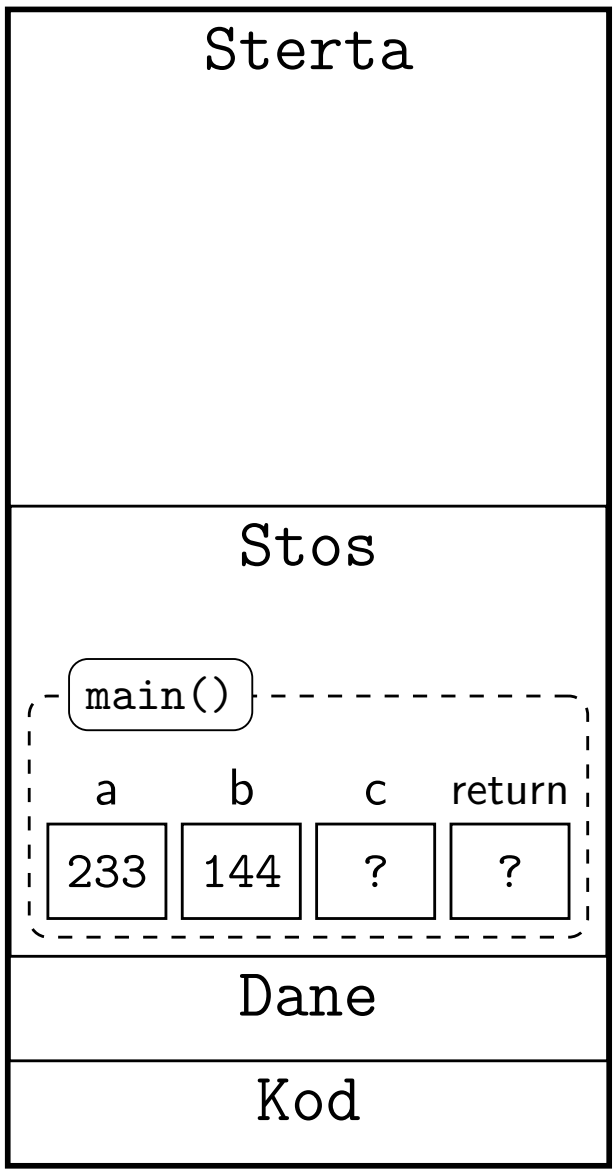
```
 1  int nwd(int a, int b)
 2  {
 3      int c;
 4      while (b != 0)
 5      {
 6          c = a % b;
 7          a = b;
 8          b = c;
 9      }
10      return a;
11  }
12
13  int main()
14  {
15      int a, b, c;
16      a = 233;
17      b = 144;
18      c = nwd(a, b);
19      return 0;
20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



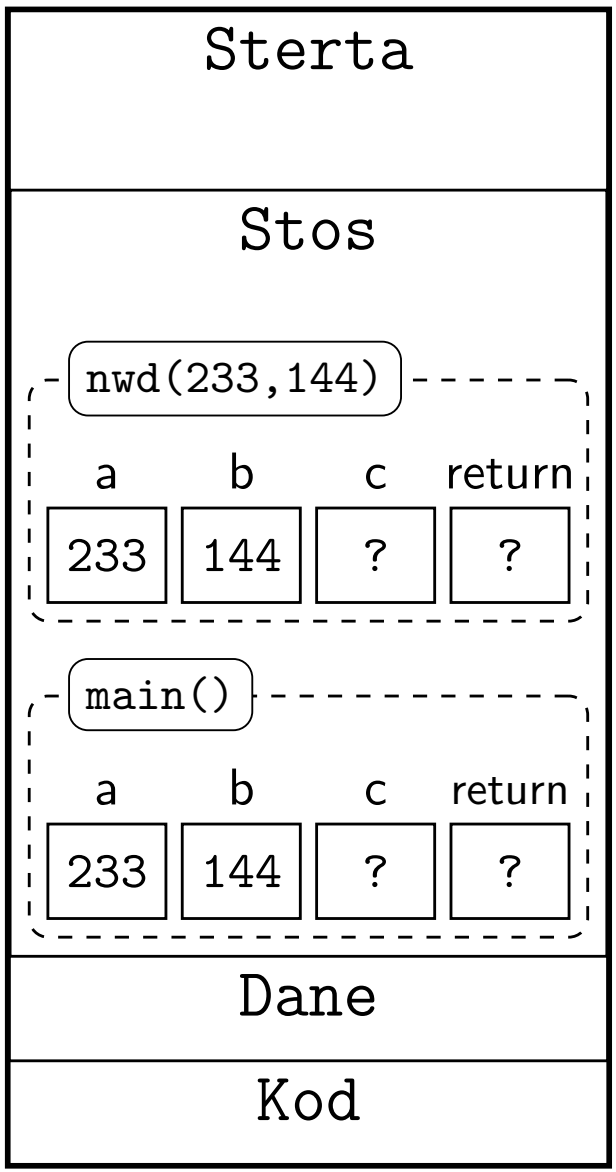
```
1  int nwd(int a, int b)
2  {
3      int c;
4      while (b != 0)
5      {
6          c = a % b;
7          a = b;
8          b = c;
9      }
10     return a;
11 }
12
13 int main()
14 {
15     int a, b, c;
16     a = 233;
17     b = 144;
18     c = nwd(a, b);
19     return 0;
20 }
```



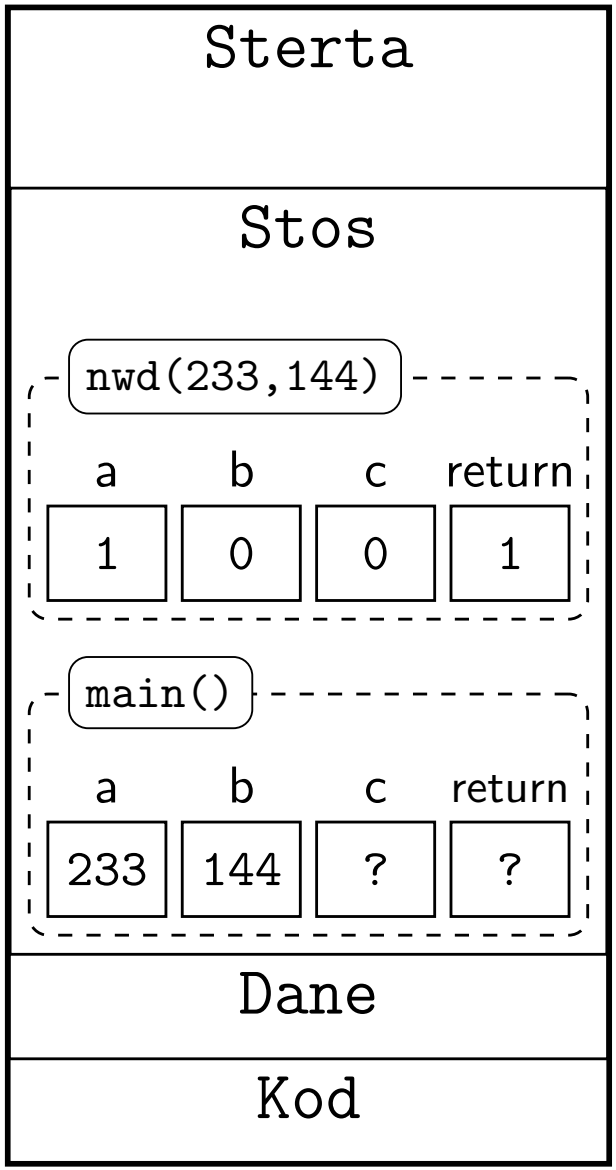
```
1  int nwd(int a, int b)
2  {
3      int c;
4      while (b != 0)
5      {
6          c = a % b;
7          a = b;
8          b = c;
9      }
10     return a;
11 }
12
13 int main()
14 {
15     int a, b, c;
16     a = 233;
17     b = 144;
18     c = nwd(a, b);
19     return 0;
20 }
```



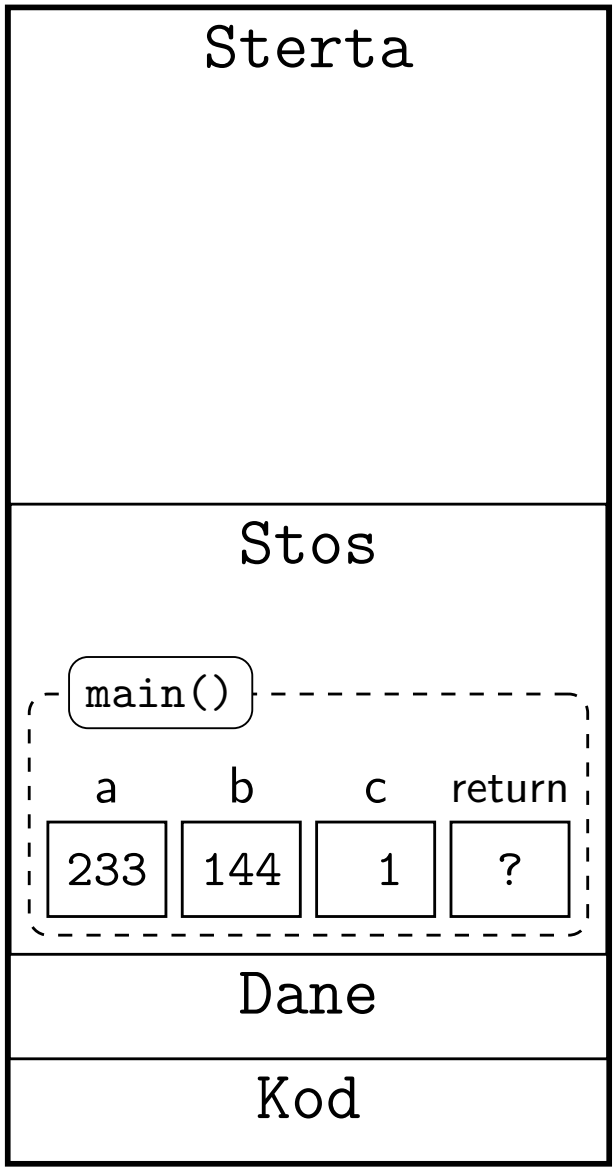
```
1  int nwd(int a, int b)
2  {
3      int c;
4      while (b != 0)
5      {
6          c = a % b;
7          a = b;
8          b = c;
9      }
10     return a;
11 }
12
13 int main()
14 {
15     int a, b, c;
16     a = 233;
17     b = 144;
18     c = nwd(a, b);
19     return 0;
20 }
```



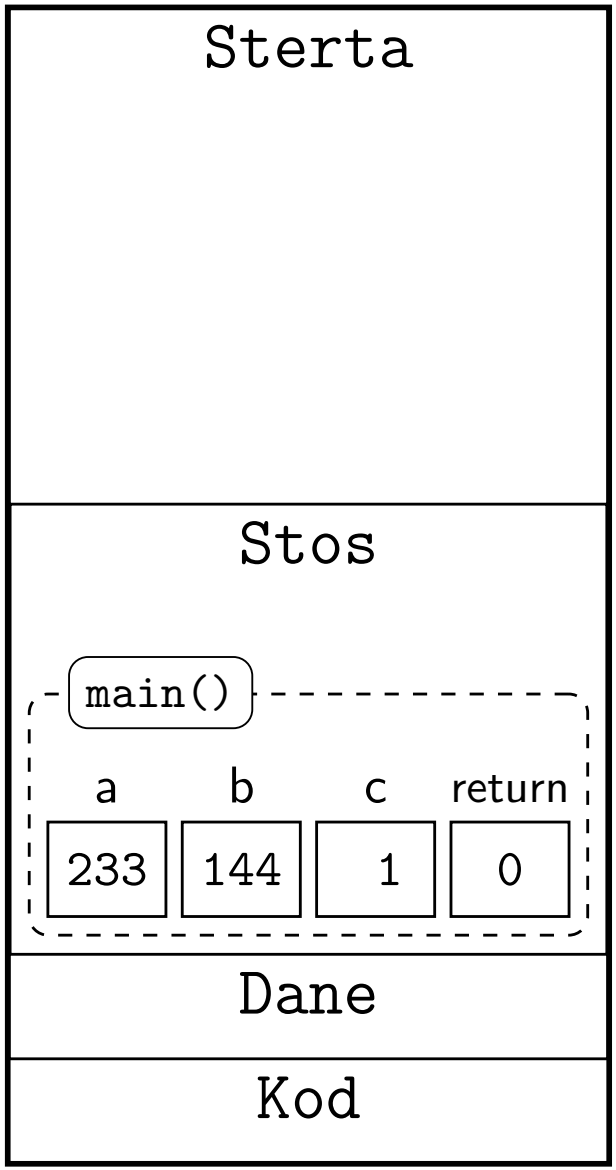
```
1  int nwd(int a, int b)
2  {
3      int c;
4      while (b != 0)
5      {
6          c = a % b;
7          a = b;
8          b = c;
9      }
10     return a;
11 }
12
13 int main()
14 {
15     int a, b, c;
16     a = 233;
17     b = 144;
18     c = nwd(a, b);
19     return 0;
20 }
```



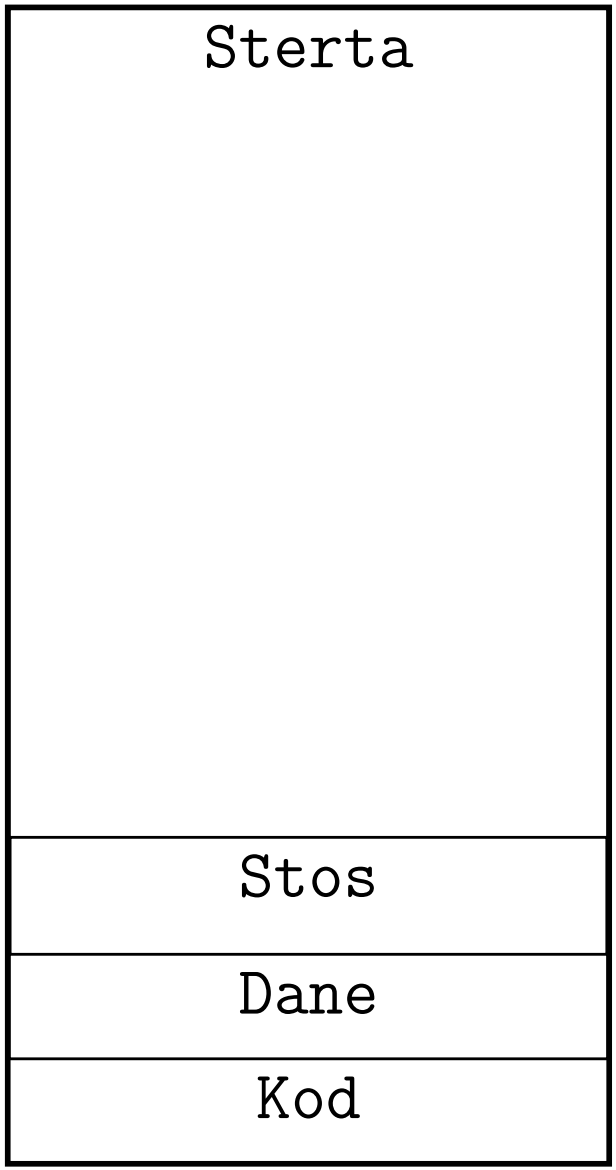
```
 1  int nwd(int a, int b)
 2  {
 3      int c;
 4      while (b != 0)
 5      {
 6          c = a % b;
 7          a = b;
 8          b = c;
 9      }
10      return a;
11  }
12
13  int main()
14  {
15      int a, b, c;
16      a = 233;
17      b = 144;
18      c = nwd(a, b);
19      return 0;
20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



```
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 jest dostępna dla wszystkich funkcji programu (ma zasięg pliku). Istnieje przez cały czas działania programu. Deklaracja zmiennej nie znajduje się wewnątrz żadnej funkcji.

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

Zmienna lokalna (automatyczna) to wewnętrzna zmienna funkcji. Czas życia zmiennej jest ograniczony czasem działania funkcji.

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

```c
#include<stdio.h>

int globalna = 0;

void f(void)
{
    int lokalna = 0;
    globalna = globalna + 1;
    lokalna = lokalna + 1;
    printf("%d          %d       f\n", globalna, lokalna);
}

int main()
{
    int lokalna = 42;

    printf("globalna lokalna funkcja\n");
    printf("%d         %d       main\n", globalna, lokalna);
    f();
    printf("%d         %d       main\n", globalna, lokalna);
    f();
    printf("%d         %d       main\n", globalna, lokalna);
    return 0;
}
```

☞ globalne.c

```c
#include<stdio.h>

int globalna = 0;

void f(void)
{
    int lokalna = 0;
    globalna = globalna + 1;
    lokalna = lokalna + 1;
    printf("%d          %d       f\n", globalna, lokalna);
}

int main()
{
    int lokalna = 42;

    printf("globalna lokalna funkcja\n");
    printf("%d          %d       main\n", globalna, lokalna);
    f();
    printf("%d          %d       main\n", globalna, lokalna);
    f();
    printf("%d          %d       main\n", globalna, lokalna);
    return 0;
}
```

Wynik działania:

```
globalna lokalna funkcja
0          42       main
1           1       f
1          42       main
2           1       f
2          42       main
```

☞ globalne.c

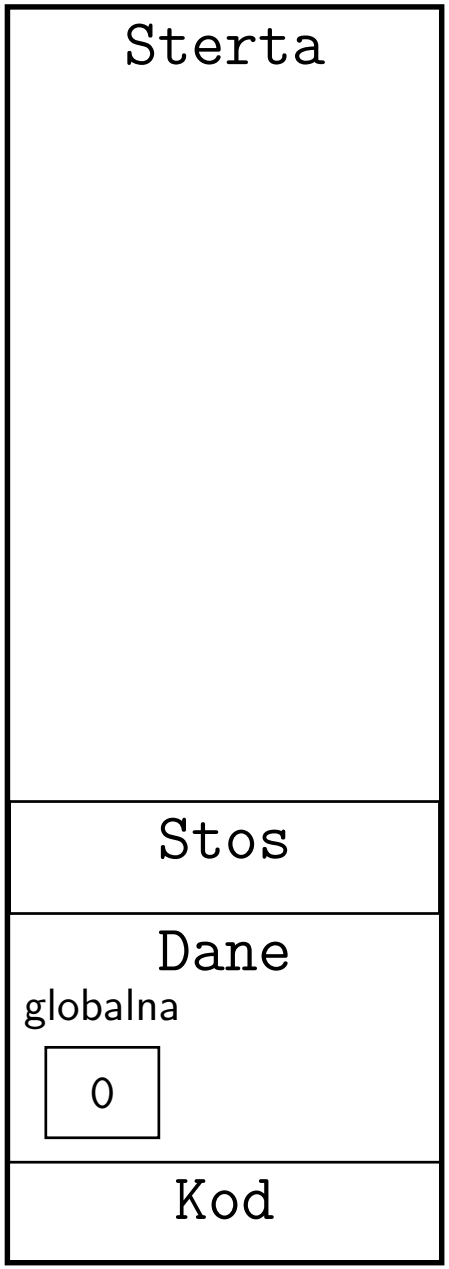
```c
#include<stdio.h>

int globalna = 0;

void f(void)
{
    int lokalna = 0;
    globalna = globalna + 1;
    lokalna = lokalna + 1;
}

int main()
{
    int lokalna = 42;
    globalna = globalna + 1;
    f();

    return 0;
}
```



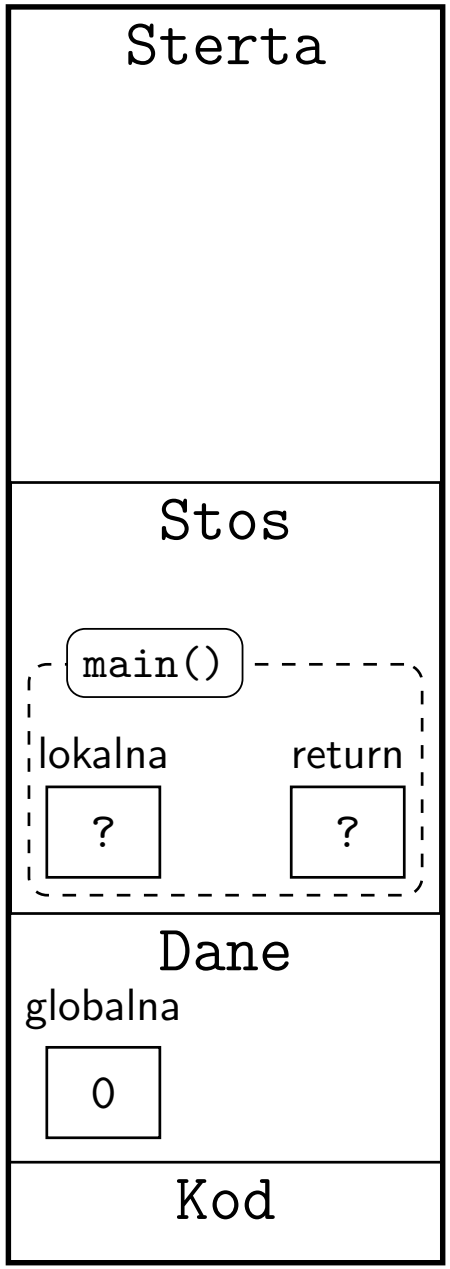
```c
#include<stdio.h>

int globalna = 0;

void f(void)
{
    int lokalna = 0;
    globalna = globalna + 1;
    lokalna = lokalna + 1;
}

int main()
{
    int lokalna = 42;
    globalna = globalna + 1;
    f();

    return 0;
}
```



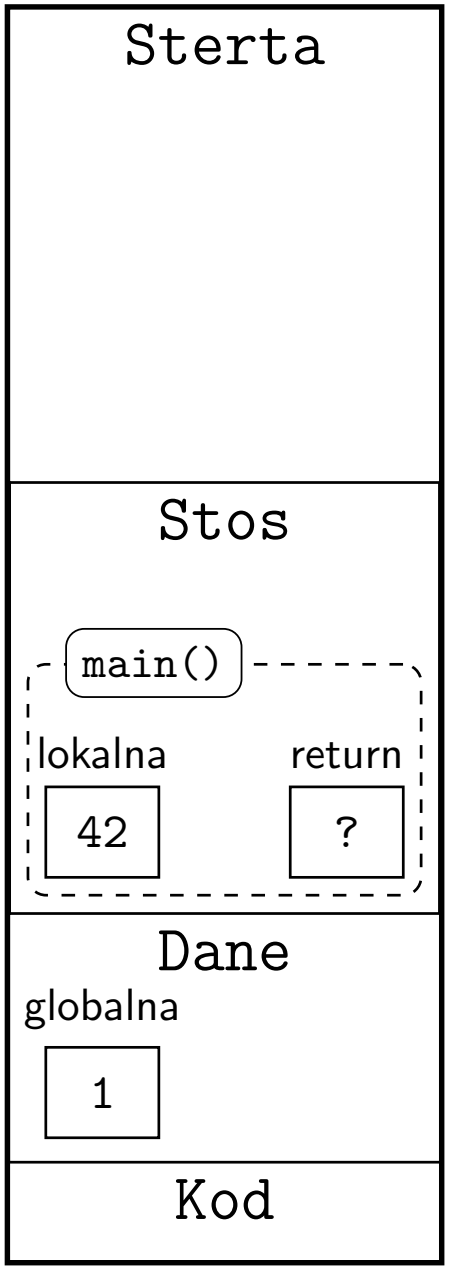
```c
#include<stdio.h>

int globalna = 0;

void f(void)
{
    int lokalna = 0;
    globalna = globalna + 1;
    lokalna = lokalna + 1;
}

int main()
{
    int lokalna = 42;
    globalna = globalna + 1;
    f();

    return 0;
}
```



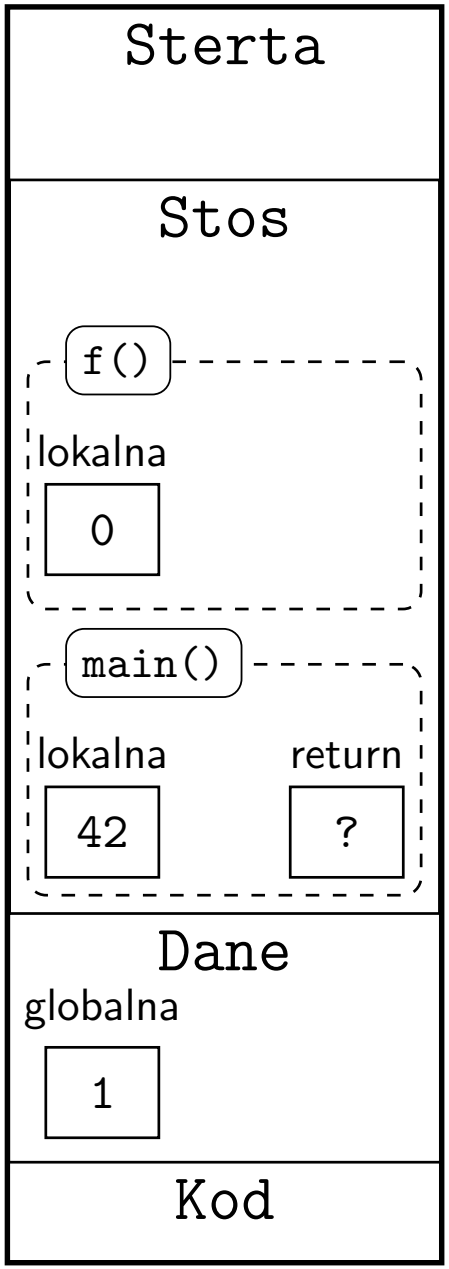
```c
#include<stdio.h>

int globalna = 0;

void f(void)
{
    int lokalna = 0;
    globalna = globalna + 1;
    lokalna = lokalna + 1;
}

int main()
{
    int lokalna = 42;
    globalna = globalna + 1;
    f();

    return 0;
}
```



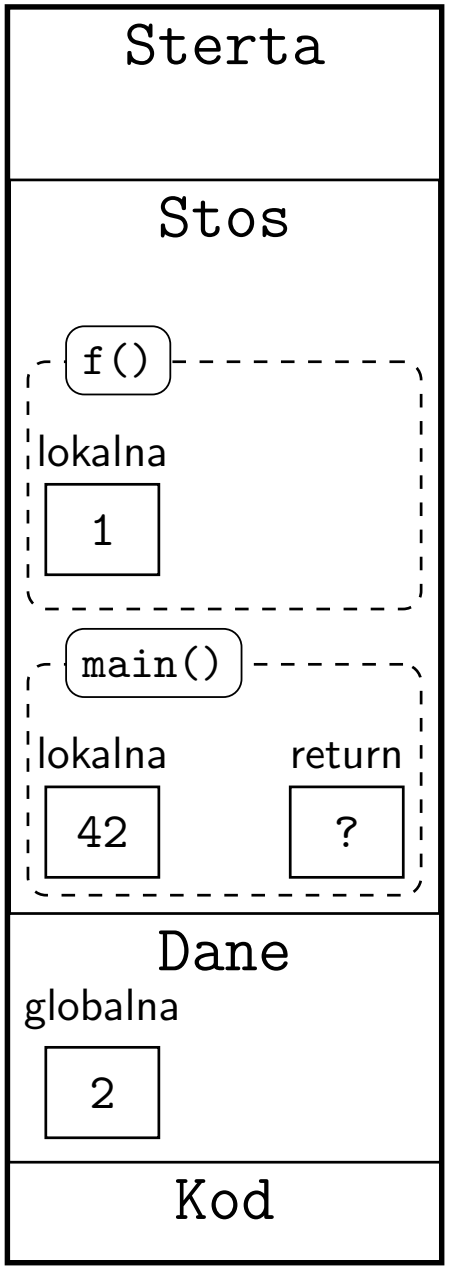
```c
#include<stdio.h>

int globalna = 0;

void f(void)
{
    int lokalna = 0;
    globalna = globalna + 1;
    lokalna = lokalna + 1;
}

int main()
{
    int lokalna = 42;
    globalna = globalna + 1;
    f();

    return 0;
}
```



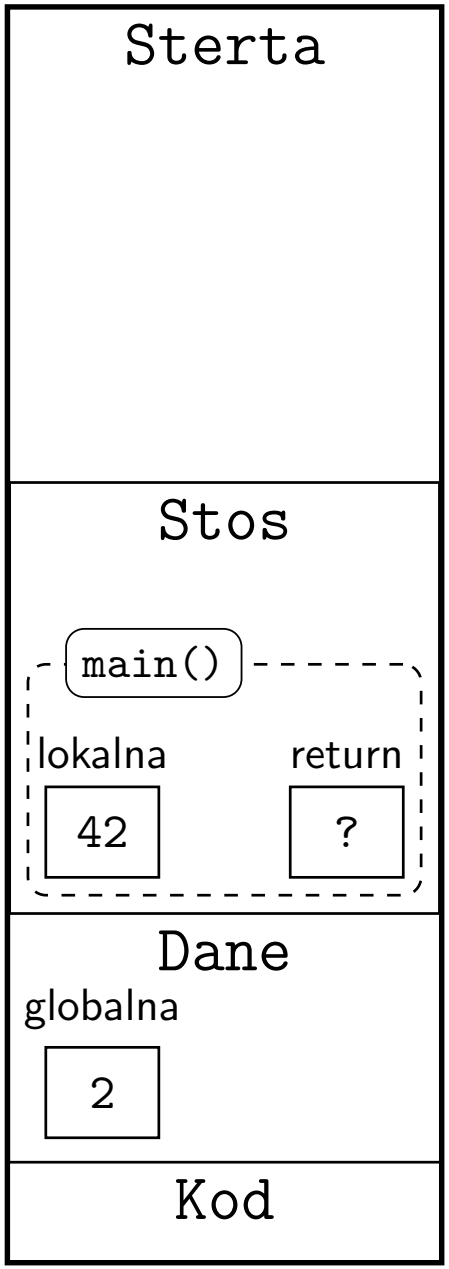
```c
#include<stdio.h>

int globalna = 0;

void f(void)
{
    int lokalna = 0;
    globalna = globalna + 1;
    lokalna = lokalna + 1;
}

int main()
{
    int lokalna = 42;
    globalna = globalna + 1;
    f();

    return 0;
}
```



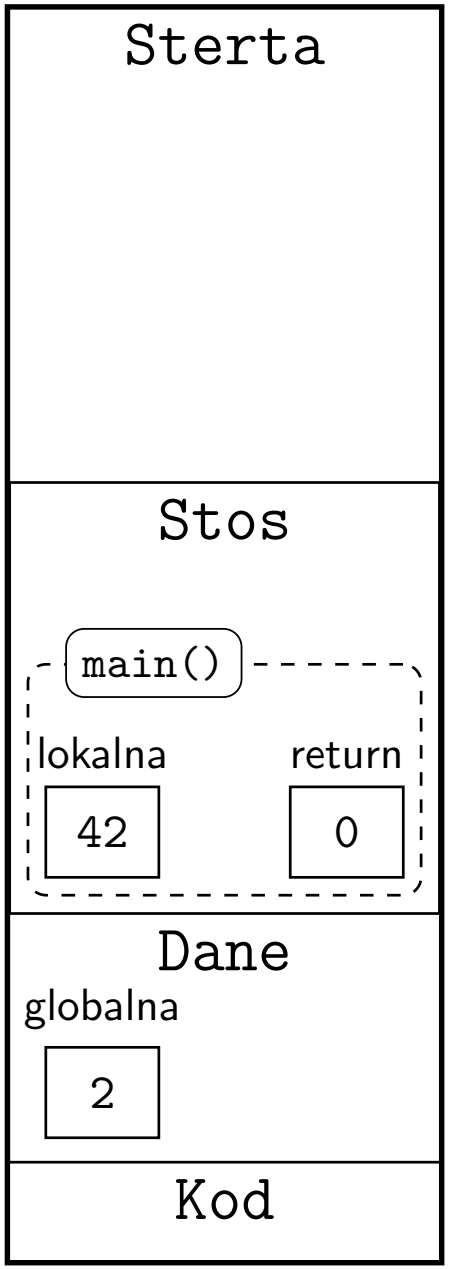
```c
#include<stdio.h>

int globalna = 0;

void f(void)
{
    int lokalna = 0;
    globalna = globalna + 1;
    lokalna = lokalna + 1;
}

int main()
{
    int lokalna = 42;
    globalna = globalna + 1;
    f();

    return 0;
}
```



W programowaniu proceduralnym nie używamy zmiennych globalnych!

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

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

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

Źle

```c
#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);
}
```

Dobrze

`<assert.h>`	asercje (do diagnostyki kodu)
`<ctype.h>`	klasyfikacja znaków
`<errno.h>`	deklaracje kodów błędów
`<float.h>`	ograniczenia typów zmiennopozycyjnych
`<limits.h>`	ograniczenia typów całkowitych
`<signal.h>`	obsługa sygnałów
`<stddef.h>`	standardowe definicje (makra)
`<stdio.h>`	operacje wejścia/wyjścia
`<math.h>`	funkcje matematyczne
`<stdlib.h>`	podstawowe narzędzia (np. `rand()`, `malloc()`)
`<string.h>`	operacje na łańcuchach znaków (napisach)
`<time.h>`	funkcje obsługi czasu
`<wchar.h>`	obsługa ciągów szerokich znaków (napisy Unicode)

☞ *WikiBook: spis plików nagłówkowych biblioteki standardowej*

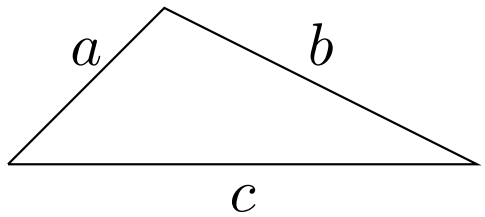
☞ *WikiBook: indeks alfabetyczny funkcji z biblioteki standardowej*

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

Uwaga: w kompilatorze GCC dołączenie funkcji z biblioteki matematycznej wymaga użycia opcji `-lm`

```
gcc -lm euclid.c
```

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

```c
int main()
{
    float a, b, c, pole;

    printf("Podaj dlugosci bokow trojkata: a, b, c > 0\n");
    scanf("%f%f%f", &a, &b, &c);

    if (a <= 0 || b <= 0 || c <= 0)
    {
        printf("Zle dane: wartosci musza byc dodatnie.\n");
        return 1;
    }

    pole = heron(a, b, c);
    if (pole < 0)
    {
       printf("Zle dane: to nie sa boki trojkata\n");
       return 2;
    }
    printf("Pole wynosi: %f\n", pole);

    return 0;
}
```

☞ heron2.c

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

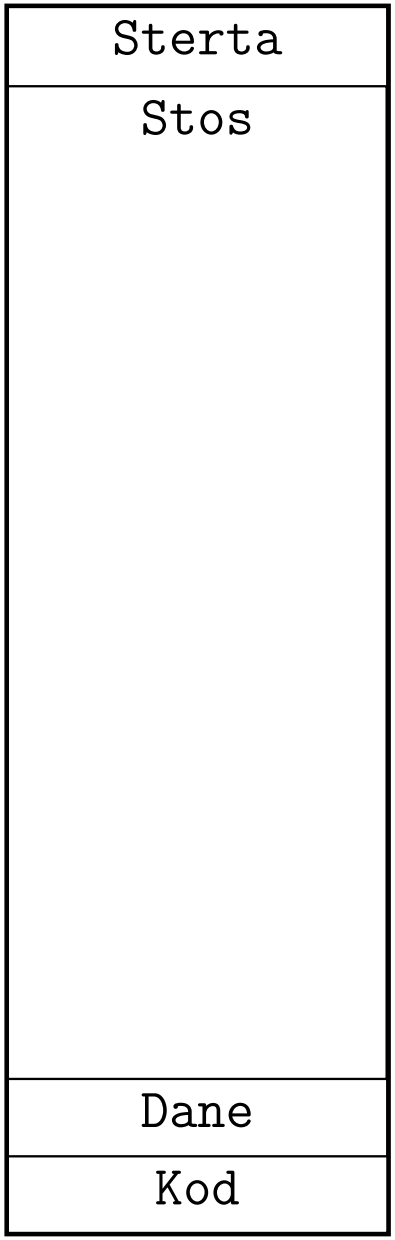
```c
#include<stdio.h>

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

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

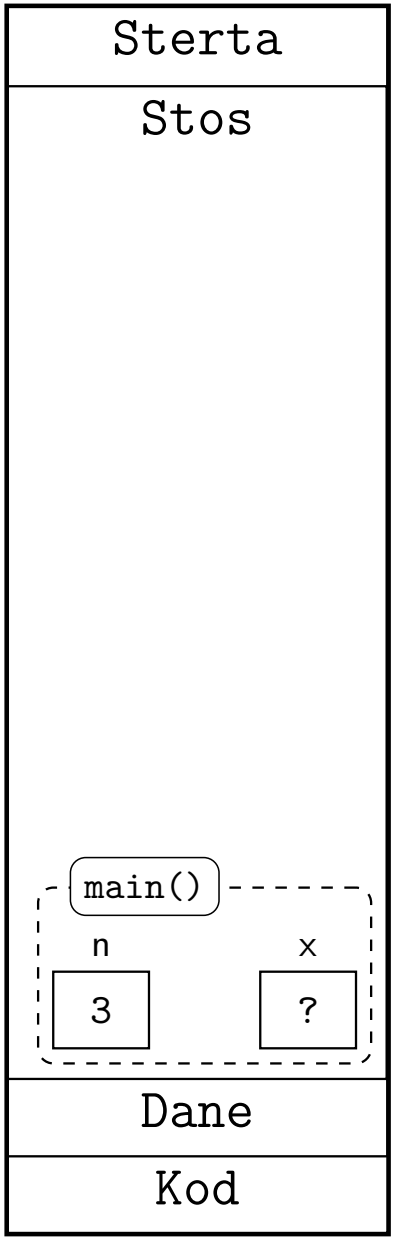
$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n = n \cdot (n-1)!$$

```
 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  }
```



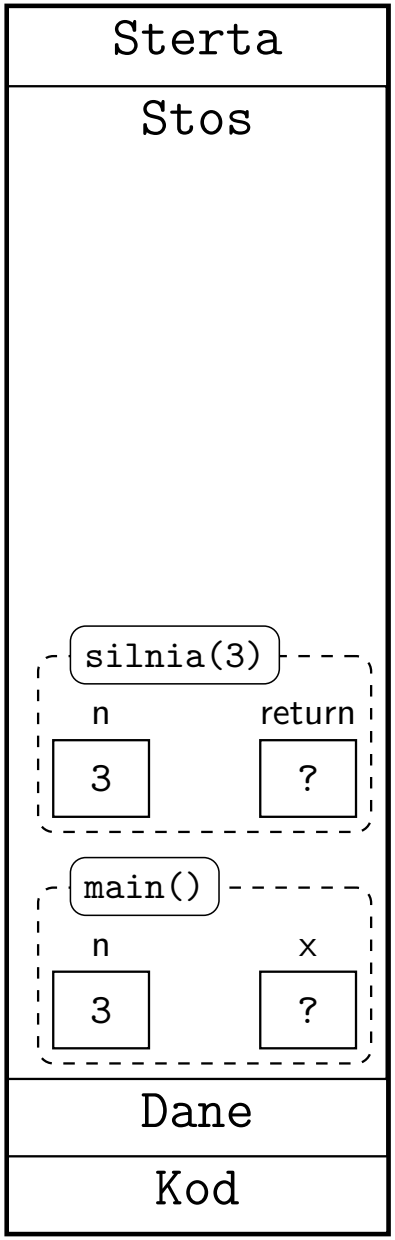
$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n = n \cdot (n-1)!$$

```
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 }
```



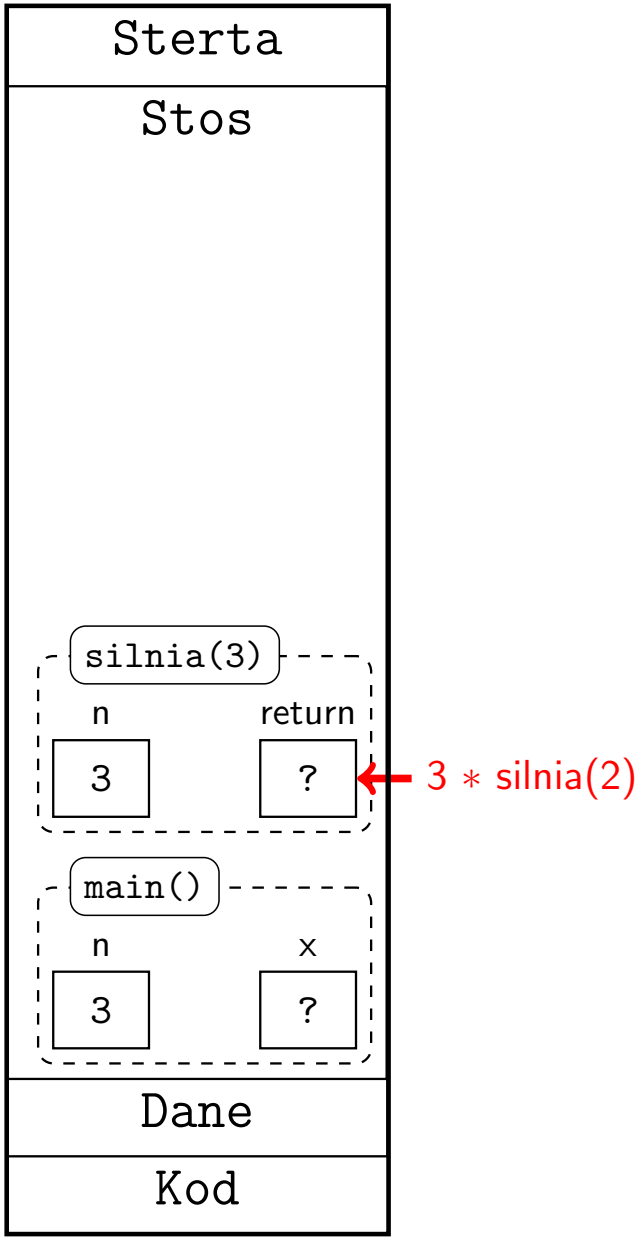
$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n = n \cdot (n-1)!$$

```
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 }
```



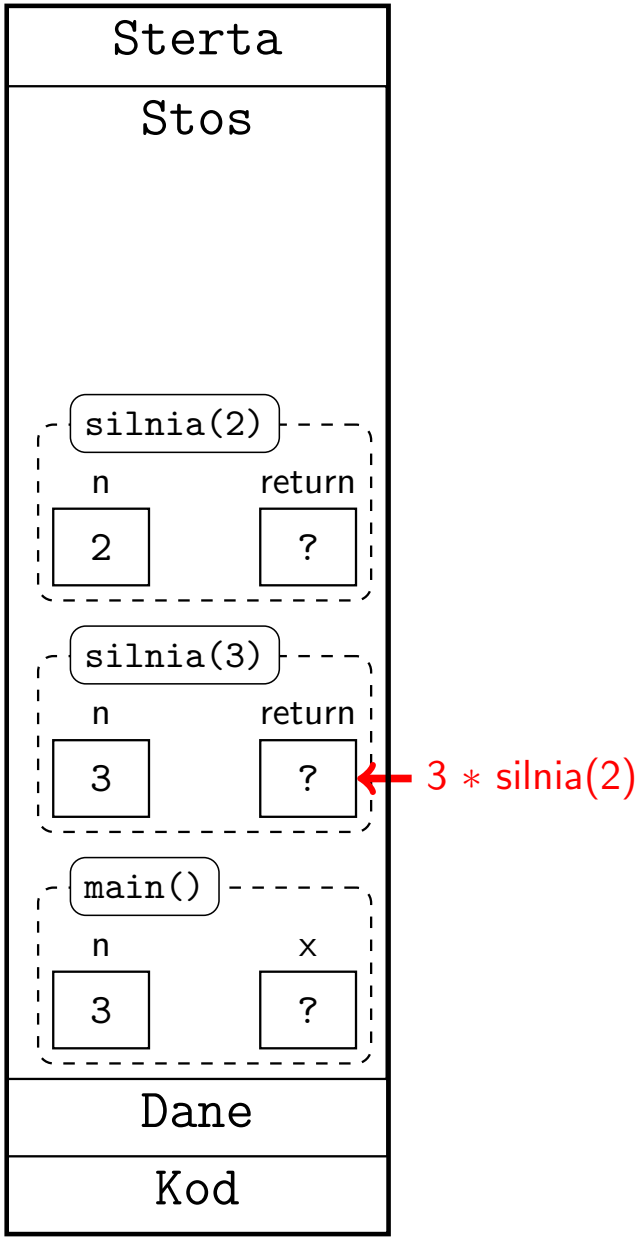
$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n = n \cdot (n-1)!$$

```
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 }
```



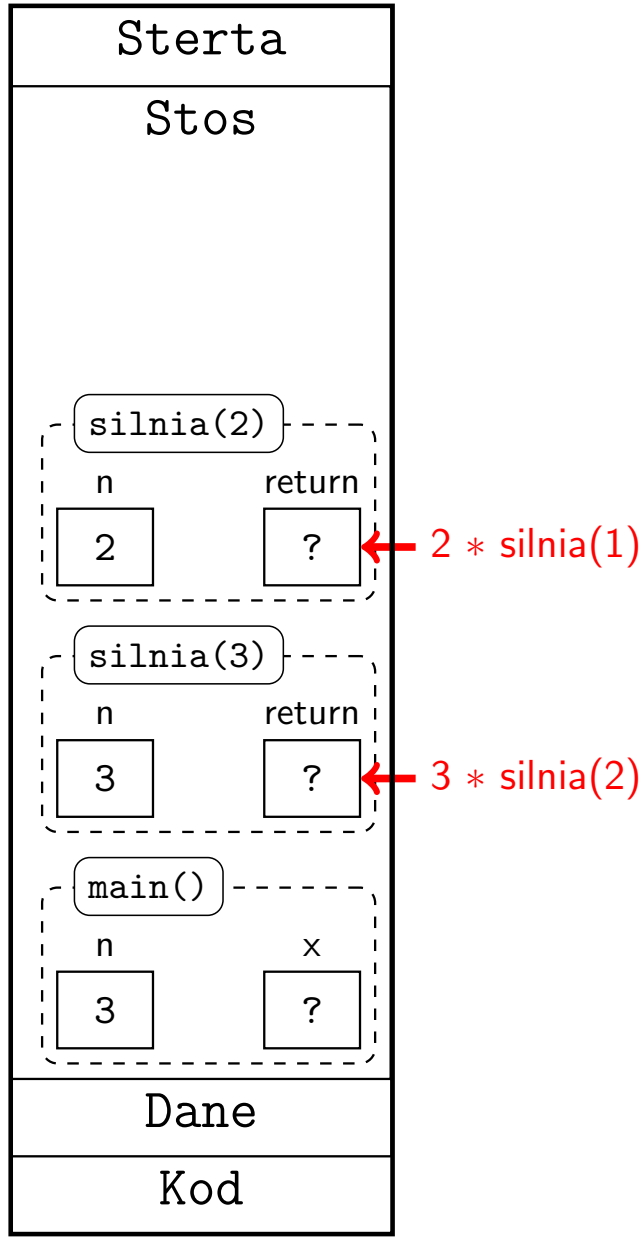
$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n = n \cdot (n-1)!$$

```
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 }
```



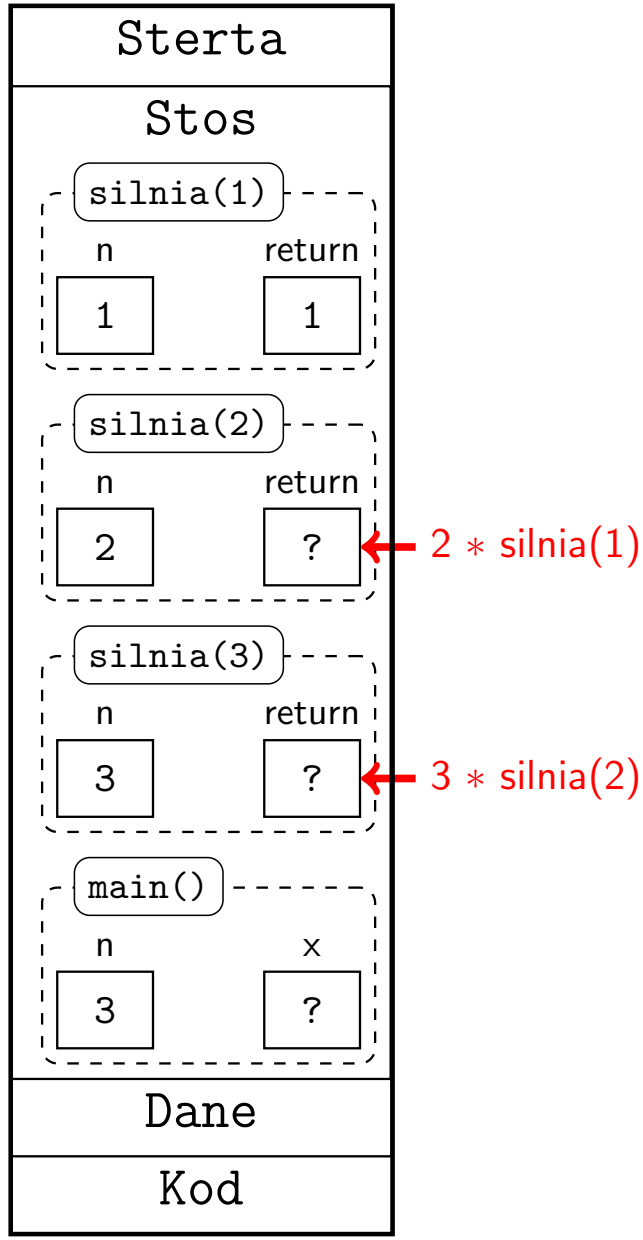
$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n = n \cdot (n-1)!$$

```
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 }
```



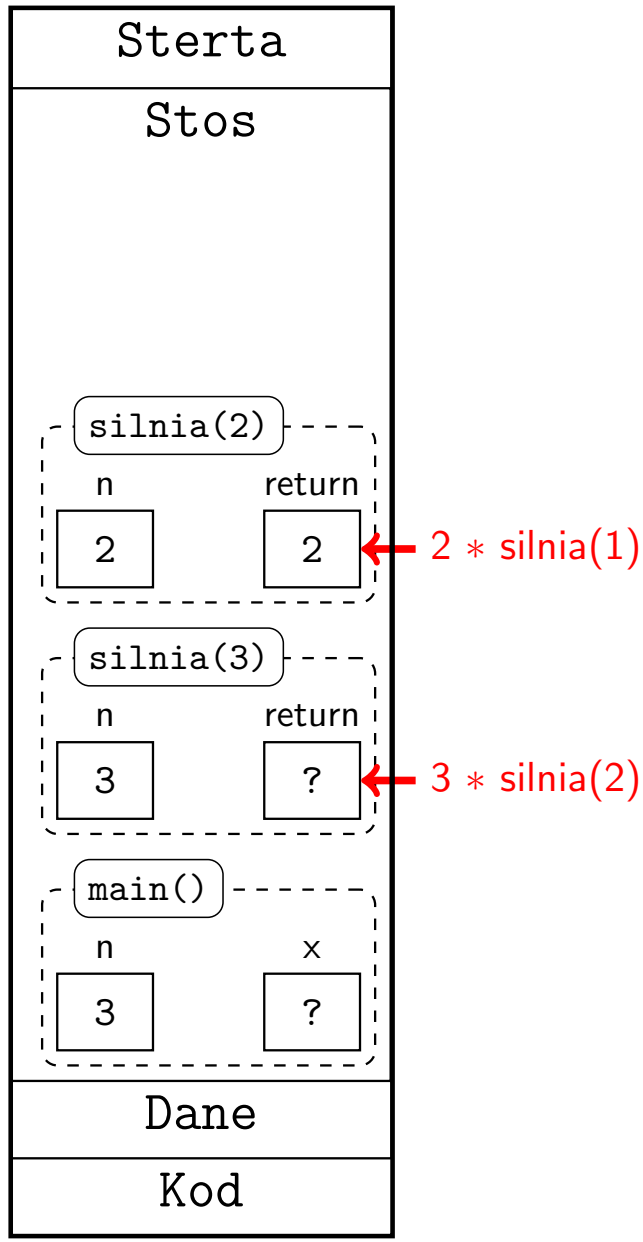
$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n = n \cdot (n-1)!$$

```
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 }
```



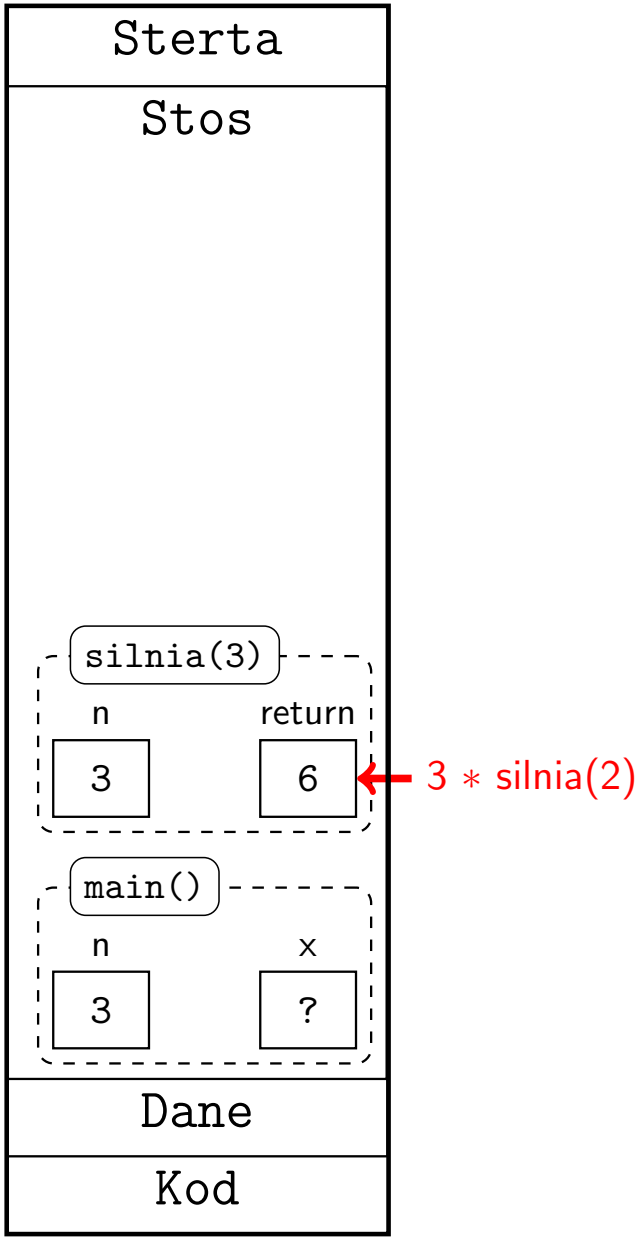
$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n = n \cdot (n-1)!$$

```
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 }
```



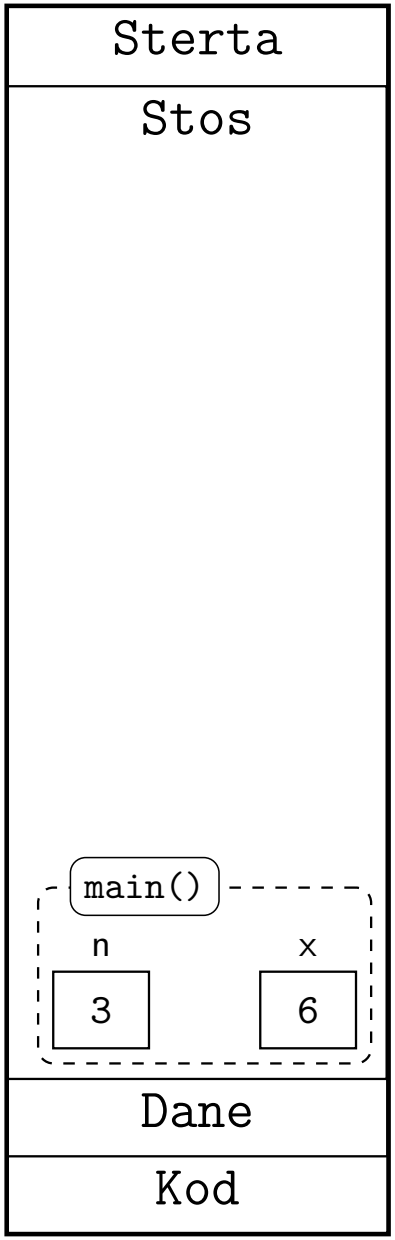
$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n = n \cdot (n-1)!$$

```
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 }
```



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

```
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 }
```



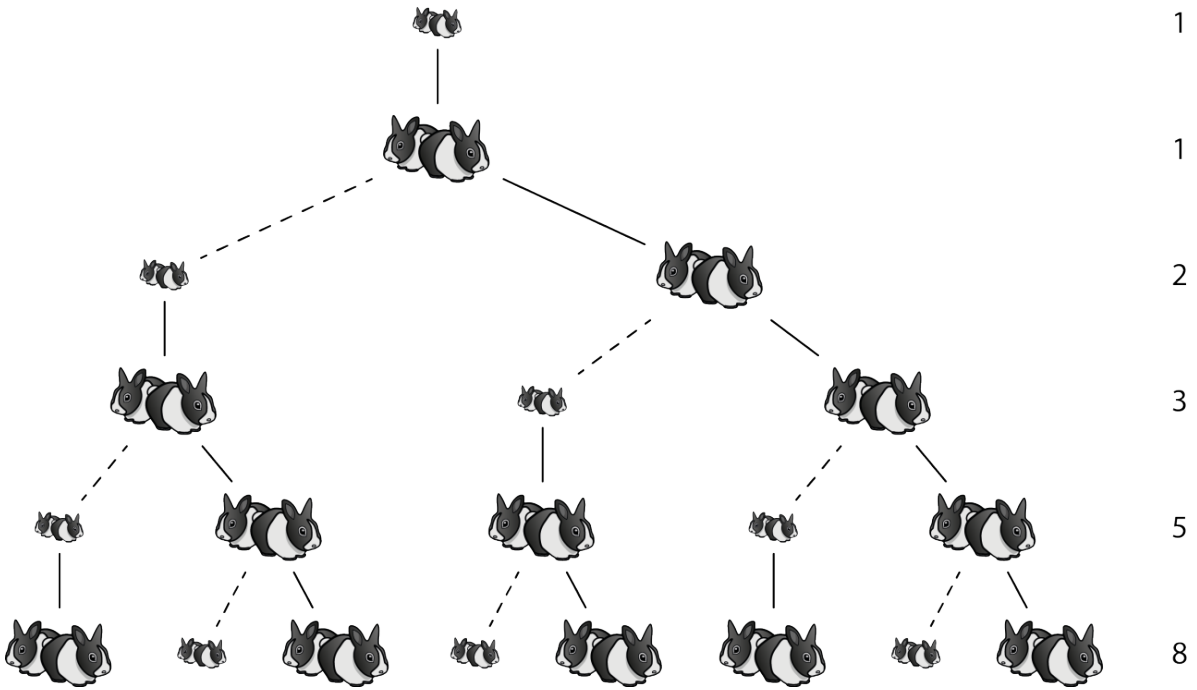
Zadanie o rozmnażaniu królików

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?

Ile par królików będzie w populacji po n miesiącach?



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

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots,$$

Ciąg Fibinacciego

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

Wersja z rekurencją

```
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

Wersja iteracyjna

```
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 dwie lub więcej wartości?
  Np. funkcja wyznaczająca miejsca zerowe paraboli powinna zwracać dwa wyniki x_1 i x_2.

Literatura dodatkowa


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`