Funkcje

czyli jak programować proceduralne.

Struktura programu w C

```c
#include <stdio.h>                          /* Dyrektywy preprocesora */
#define PI 3.1415

float g = 2.5;                              /* Zmienne globalne */

float kwadrat(float x)                      /* Definicje funkcji */
{
   return x*x;
}

int main()                                  /* Funkcja glowna */
{
    float x, y;                             /* Zmienne lokalne */

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

    return 0;
}
```

Funkcje = podprogramy

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
- Nic za darmo - wywołanie funkcji to dodatkowy koszt czasu i pamięci
- Rekurencja(Rekurencja(Rekurencja(Rekurencja(...))))

Deklaracja funkcji (prototyp)

Zapowiedź użycia funkcji - określenie nazwy funkcji i typów

```
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);
```

```c
#include <stdio.h>

int nwd(int a, int b)
{
    int c;
    while (b != 0)
    {
        c = a % b;
        a = b;
        b = c;
    }
    return a;
}

int main()
{
    int a, b;

    printf("Podaj dwie liczby calkowite: ");
    scanf("%d %d", &a, &b);
    printf("NWD(%d,%d) = %d\n", a, b, nwd(a,b));

    return 0;
}
```

☞ euclid3.c

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 !

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;
}
```

Instrukcja return

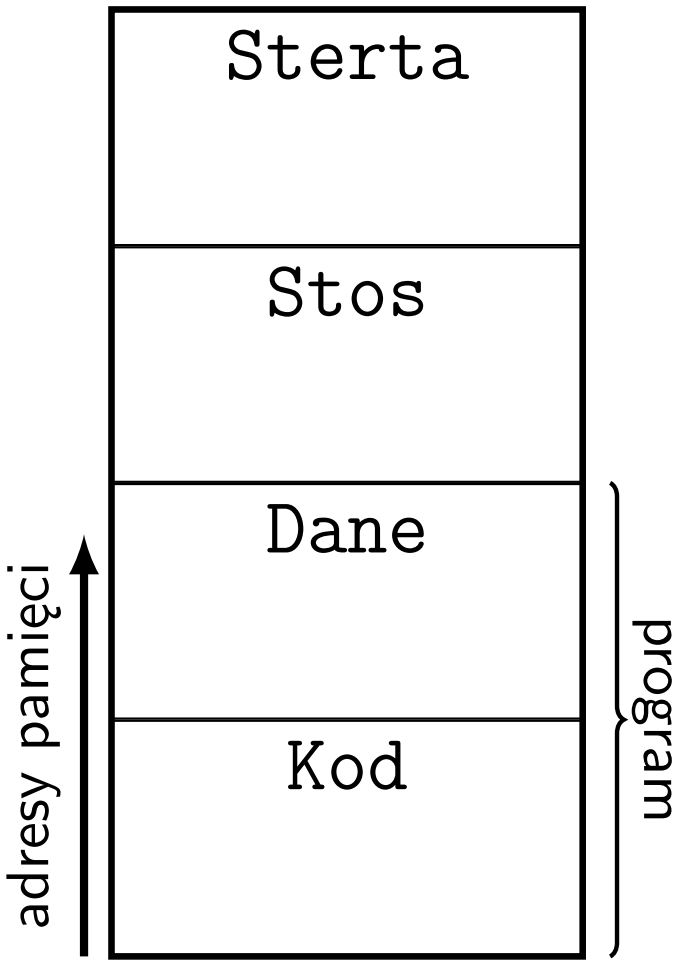
- Instrukcja `return` może pojawić się w dowolnym miejscu wewnątrz funkcji
- Wartość zwracana a typ deklarowany

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

Wartość zwracana jest podstawiona w miejsce wywołania

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

Pamięć programu



pamięć dostępna do alokacji

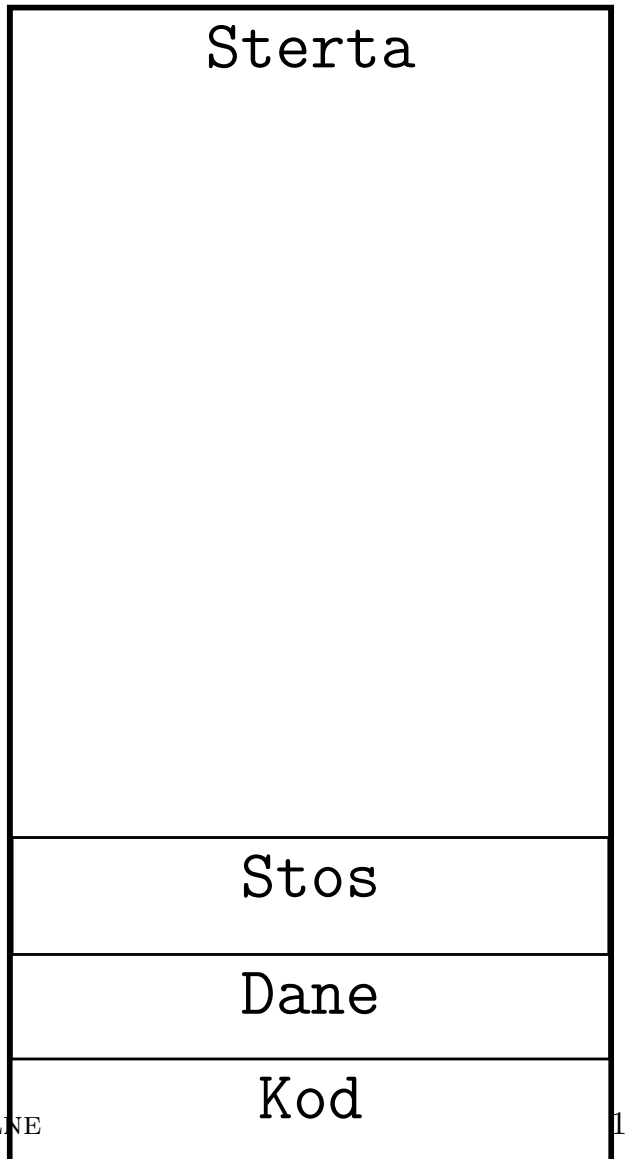
zmienne lokalne funkcji wkładane na stos przy wywołaniu funkcji

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

tekst programu
tylko do odczytu

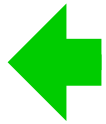
Zmienne na stosie

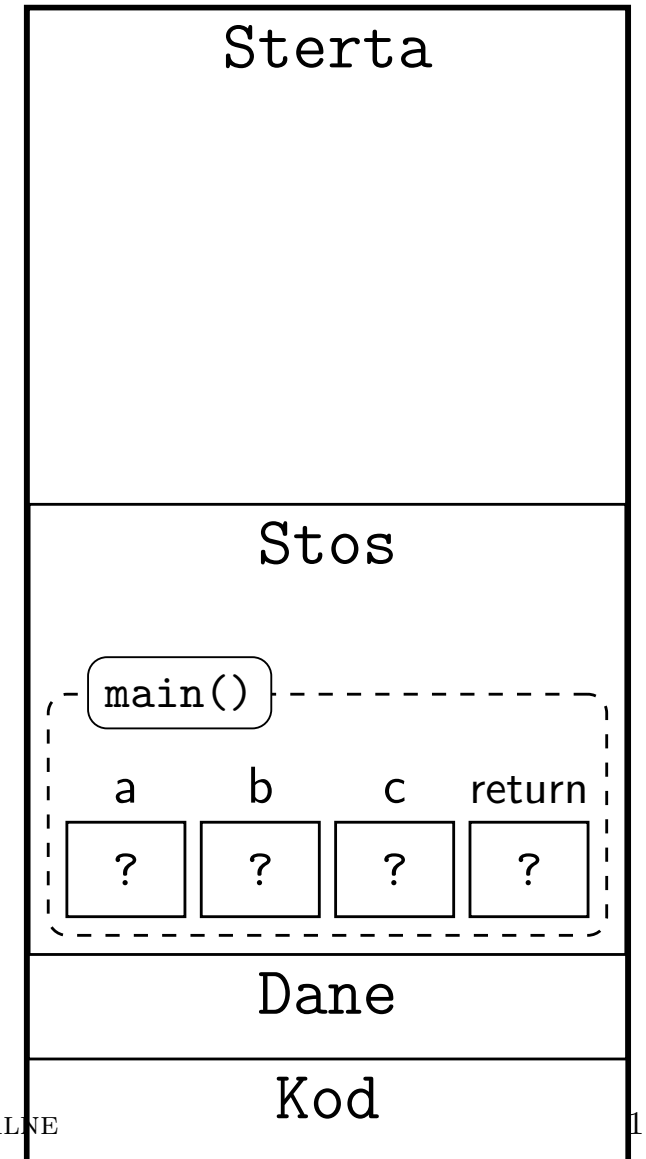
```
 1  int nwd(int a, int b)
 2  {
 3      int c;
 4      while (b != 0)
 5      {
 6          c = a % b;
 7          a = b;
 8          b = c;
 9      }
10      return a;
11  }
12
13  int main()
14  {
15      int a, b, c;
16      a = 233;
17      b = 144;
18      c = nwd(a, b);
19      return 0;
20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



Zmienne na stosie

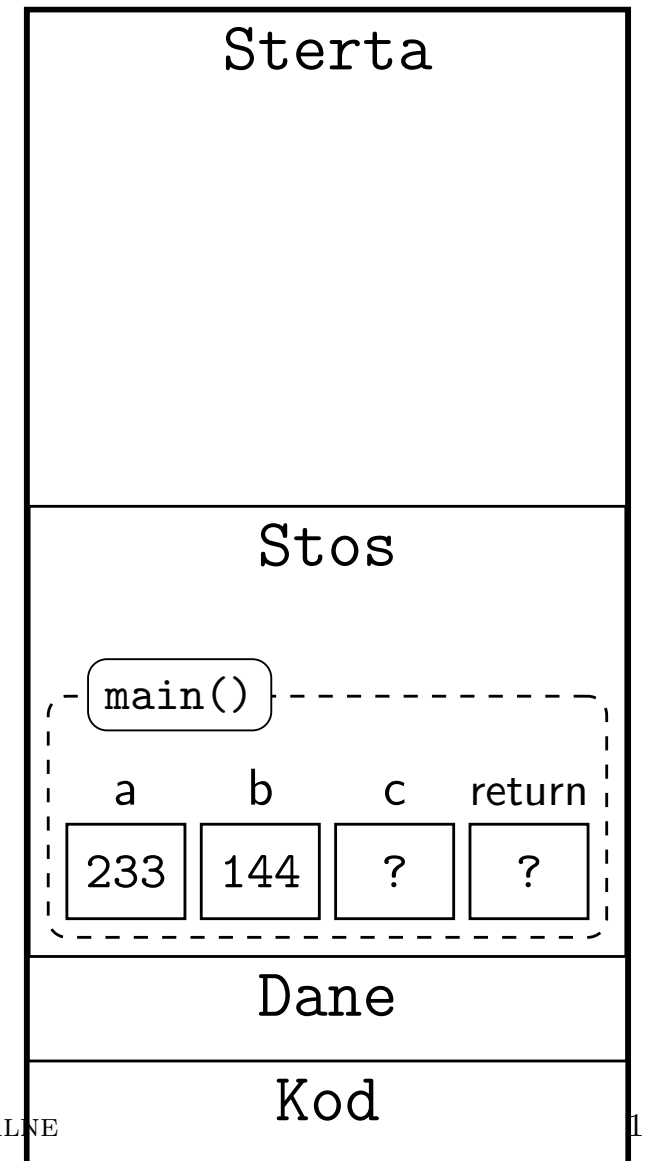
```
1  int nwd(int a, int b)
2  {
3      int c;
4      while (b != 0)
5      {
6          c = a % b;
7          a = b;
8          b = c;
9      }
10     return a;
11 }
12
13 int main()
14 {
15     int a, b, c;
16     a = 233;
17     b = 144;
18     c = nwd(a, b);
19     return 0;
20 }
```





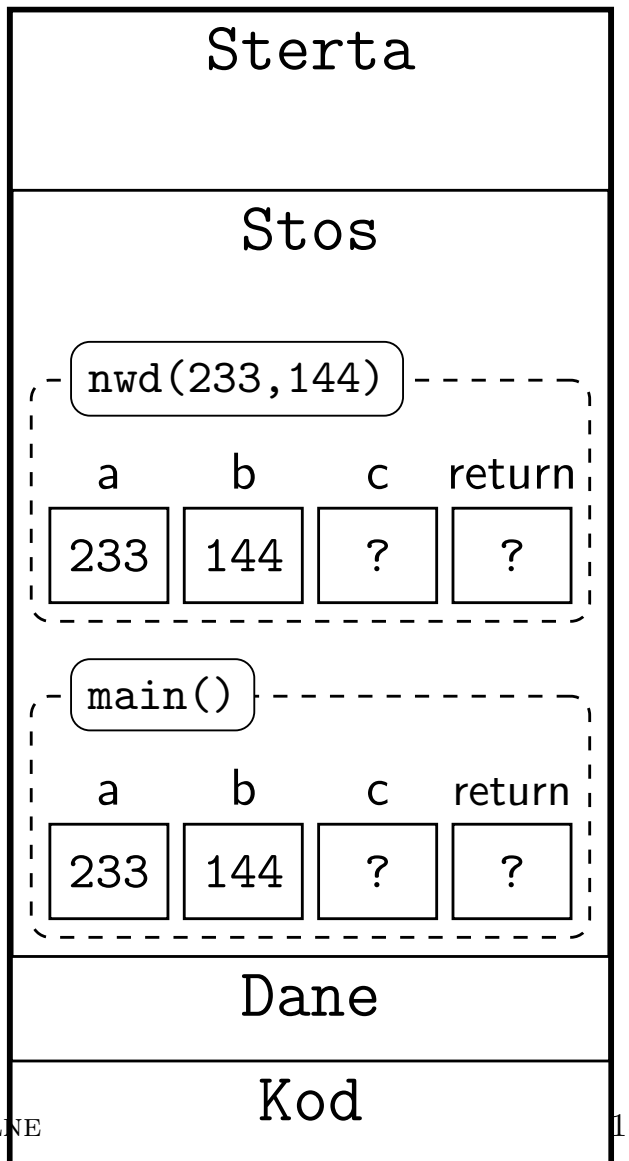
Zmienne na stosie

```
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 }
```



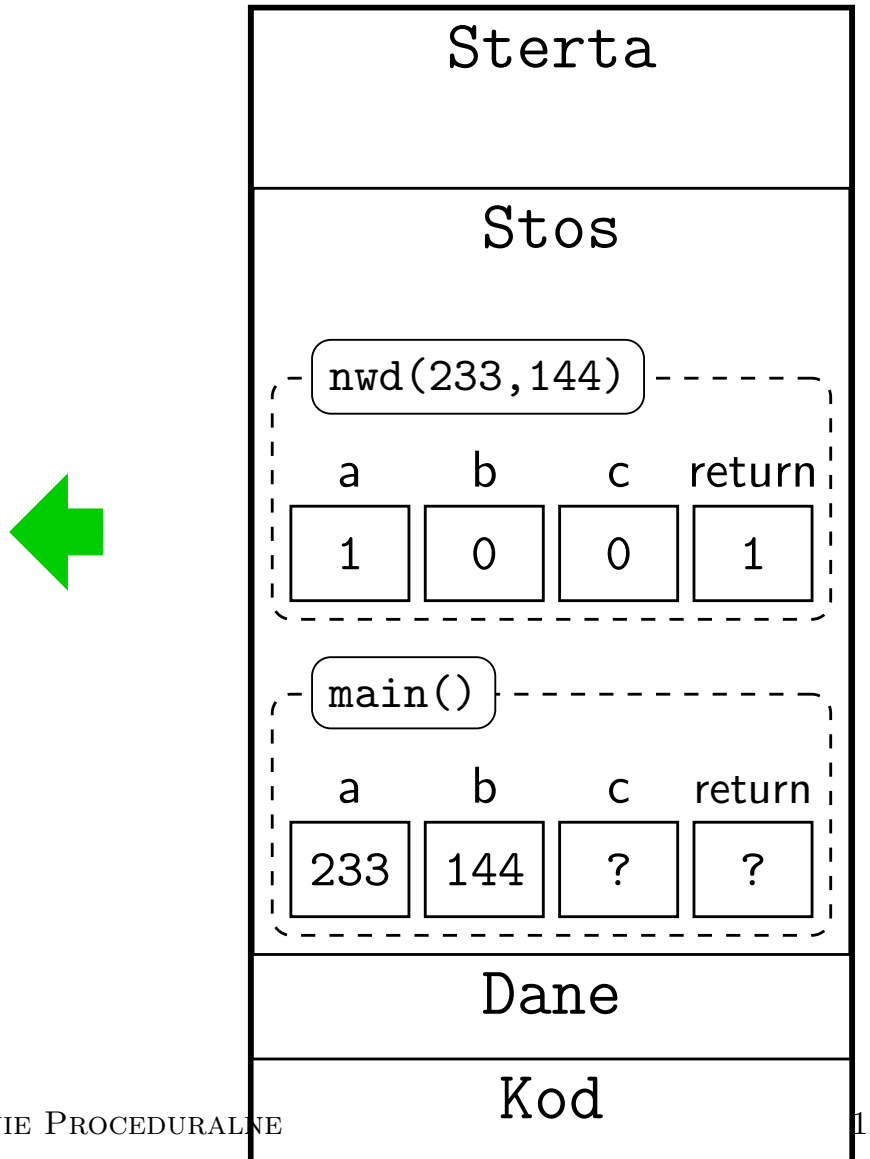
Zmienne na stosie

```
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 }
```



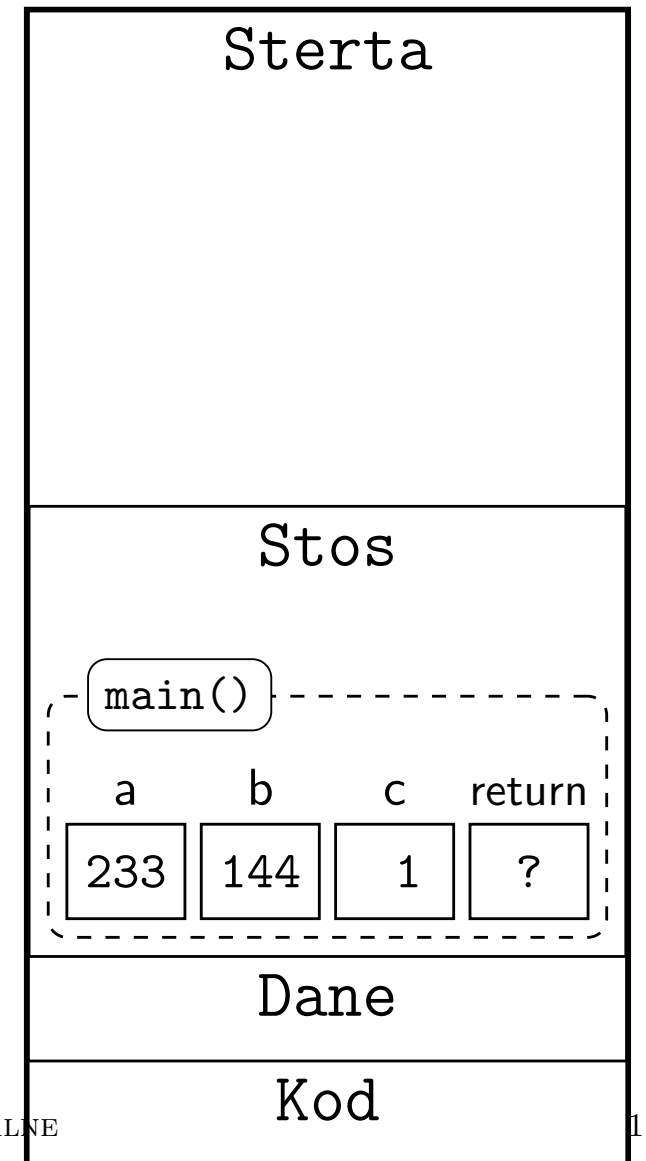
Zmienne na stosie

```
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 }
```



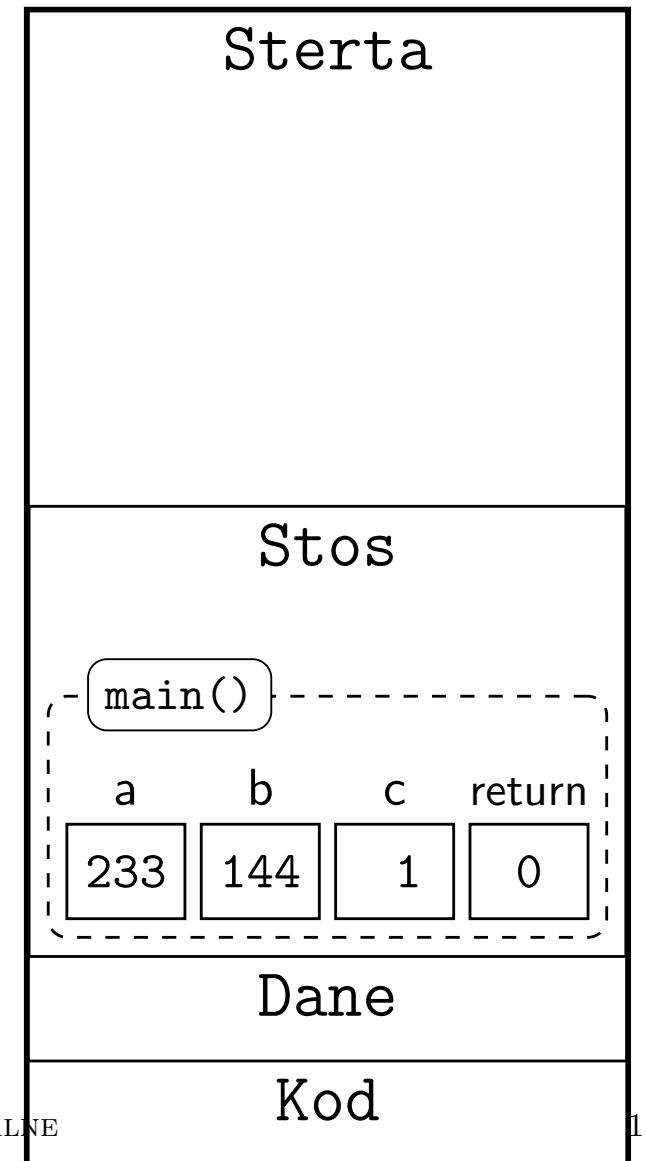
Zmienne na stosie

```
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 }
```



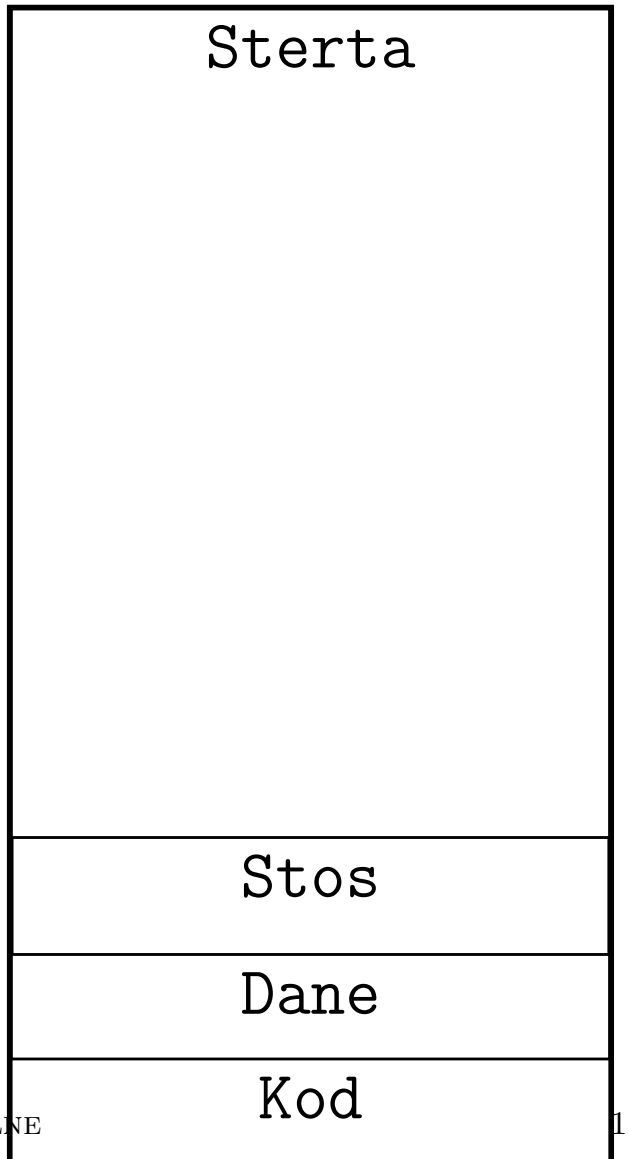
Zmienne na stosie

```
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 }
```



Zmienne na stosie

```
 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.

```
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;
}
```

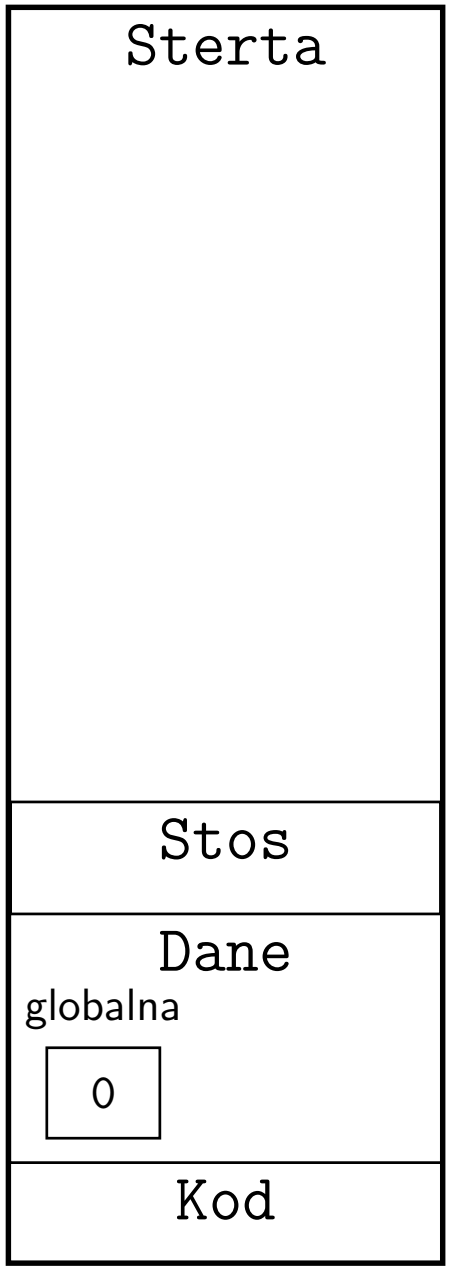
```c
#include<stdio.h>

int globalna = 0;

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

int main()
{
    int lokalna = 13;
    globalna = globalna + 1;
    f();

    return 0;
}
```



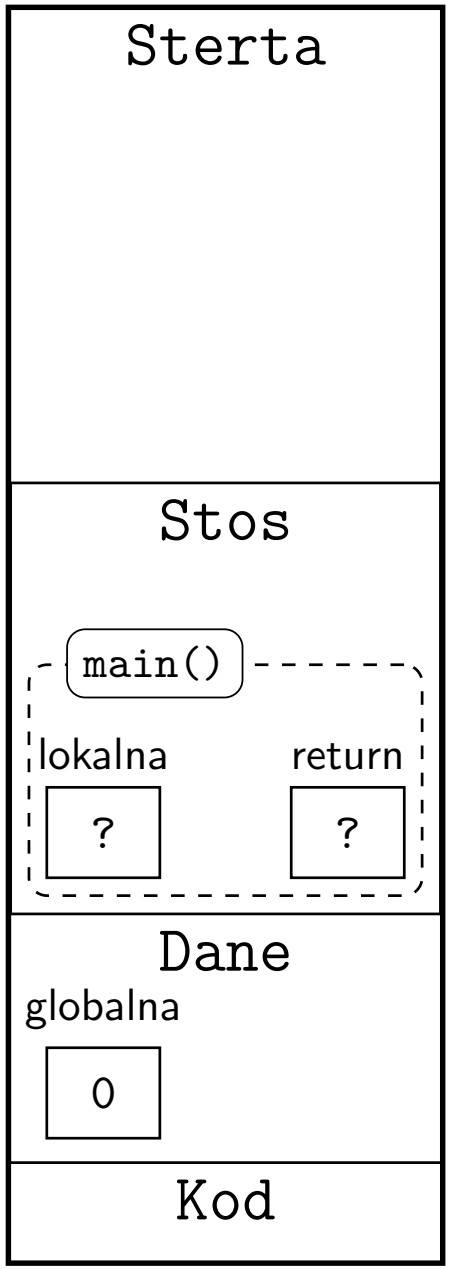
```c
#include<stdio.h>

int globalna = 0;

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

int main()
{
    int lokalna = 13;
    globalna = globalna + 1;
    f();

    return 0;
}
```



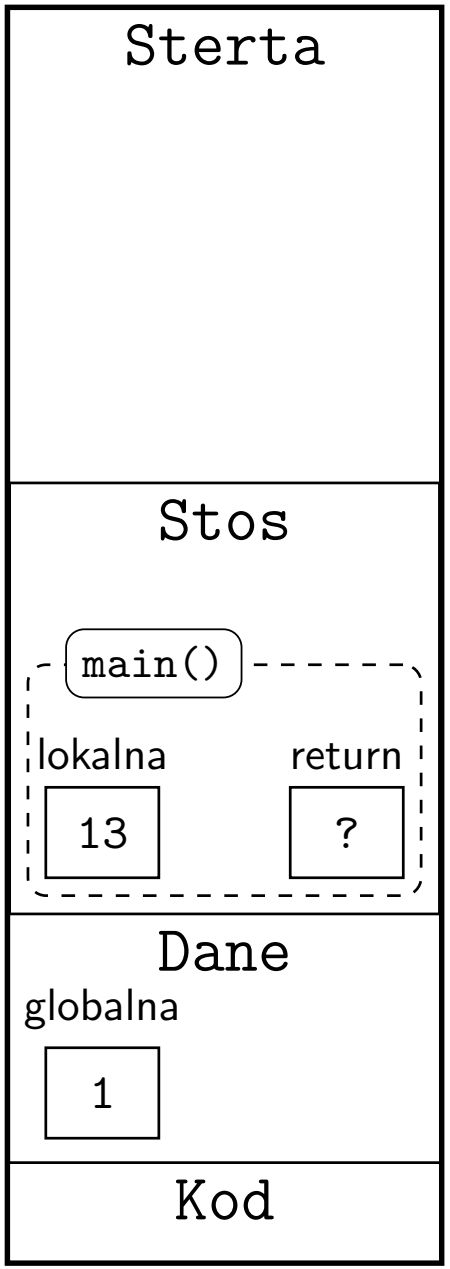
```c
#include<stdio.h>

int globalna = 0;

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

int main()
{
    int lokalna = 13;
    globalna = globalna + 1;
    f();

    return 0;
}
```



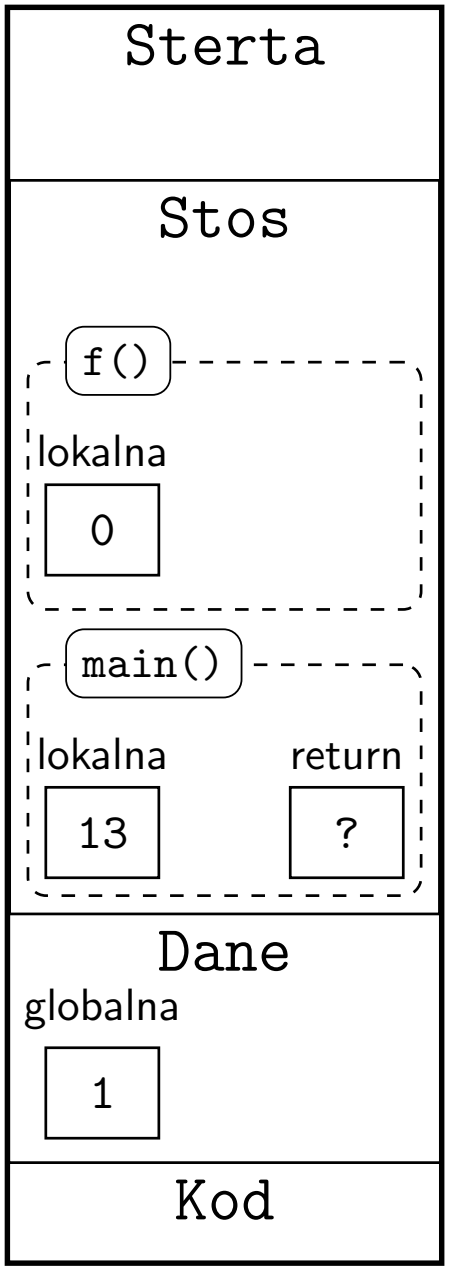
```c
#include<stdio.h>

int globalna = 0;

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

int main()
{
    int lokalna = 13;
    globalna = globalna + 1;
    f();

    return 0;
}
```



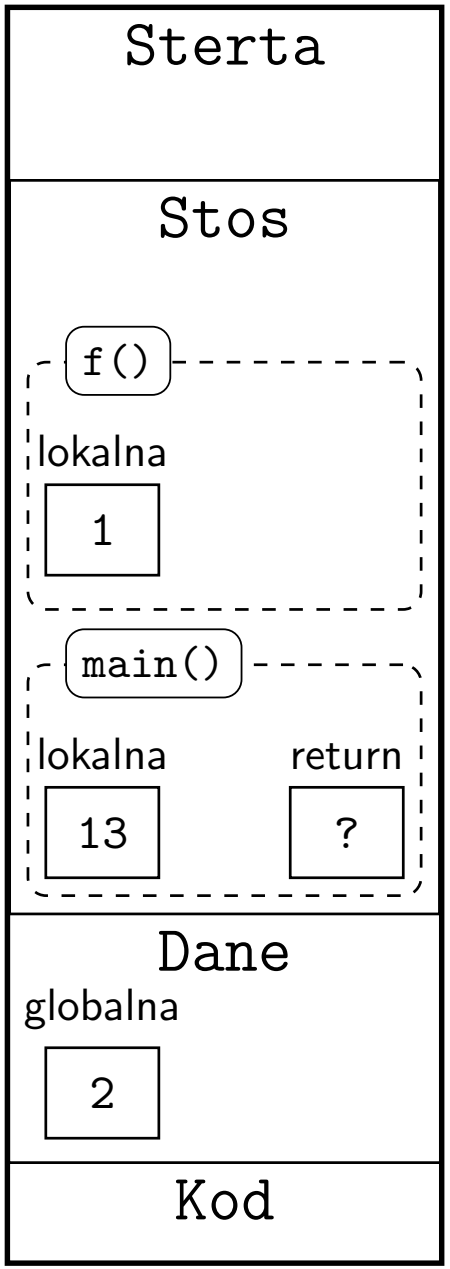
```c
#include<stdio.h>

int globalna = 0;

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

int main()
{
    int lokalna = 13;
    globalna = globalna + 1;
    f();

    return 0;
}
```



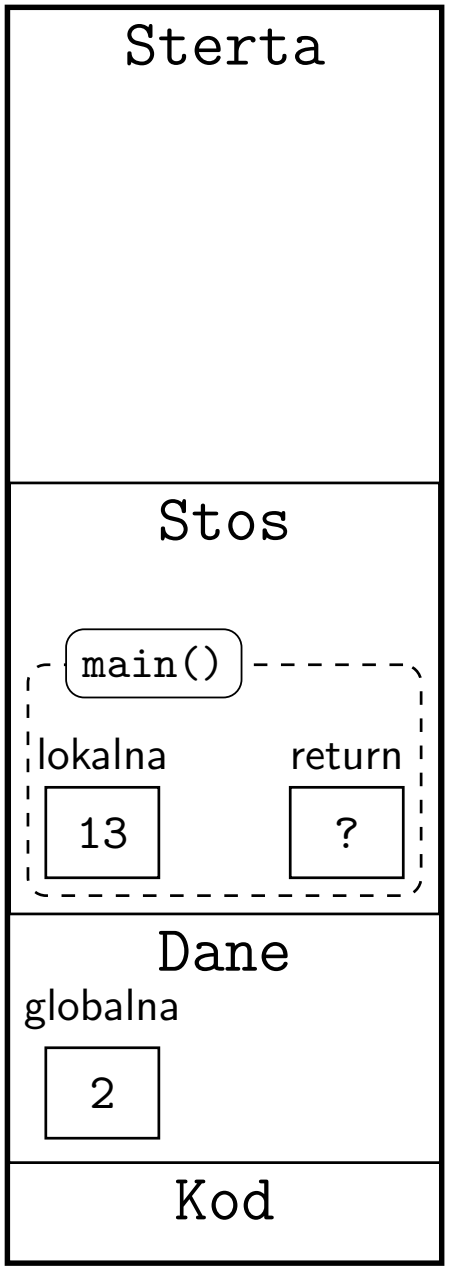
```c
#include<stdio.h>

int globalna = 0;

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

int main()
{
    int lokalna = 13;
    globalna = globalna + 1;
    f();

    return 0;
}
```



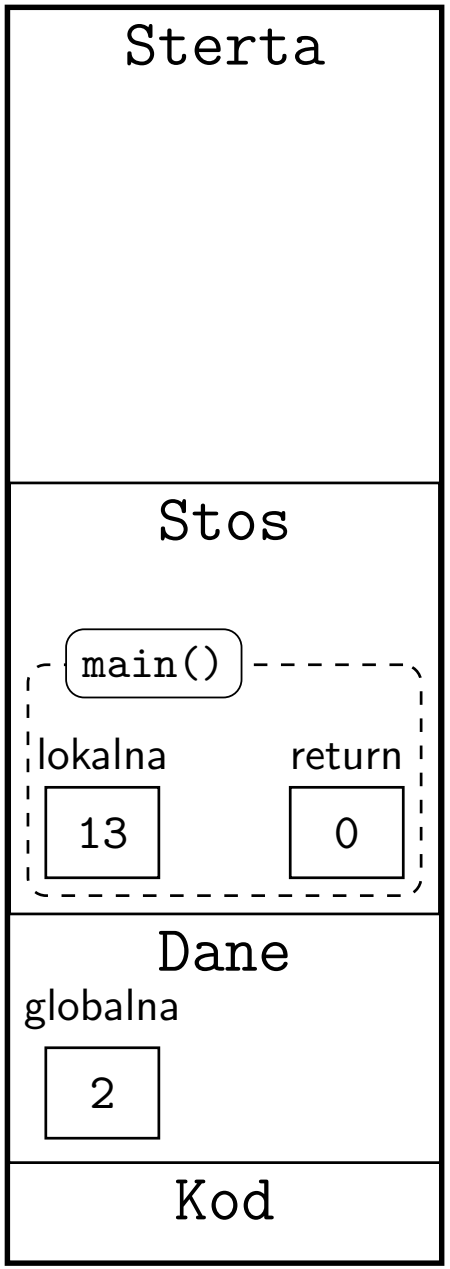
```c
#include<stdio.h>

int globalna = 0;

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

int main()
{
    int lokalna = 13;
    globalna = globalna + 1;
    f();

    return 0;
}
```



W programowaniu proceduralnym nie używamy zmiennych globalnych!

Źle

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

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

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

Dobrze

```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);
}
```

Biblioteki standardowe

`<assert.h>`	Asercje - diagnostyka kodu
`<ctype.h>`	Klasyfikacja znaków
`<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>`	Zestaw podstawowych narzędzi, np. `rand()`
`<string.h>`	Obsługa łańcuchów znakowych
`<time.h>`	Funkcje obsługi czasu

☞ *WikiBook: spis plików nagłówkowych biblioteki standardowej*

☞ *WikiBook: indeks alfabetyczny funkcji z biblioteki standardowej*

Biblioteka matematyczna

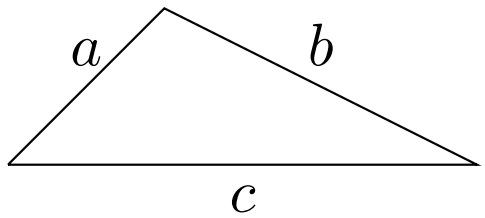
`sin`	`cos`	`tan`	funkcje trygonometryczne
`asin`	`acos`	`atan`	
`sinh`	`cosh`	`tanh`	funkcje hiperboliczne
`exp`			f. eksponencjalna e^x
`ceil`	`floor`		zaokrąglenia: *sufit*, *podłoga*
`sqrt`			pierwiastek $\sqrt{x}$
`pow`			potęga x^y
`log`			logarytm naturalny $\ln(x) = \log_e x$
`log10`			logarytm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

```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

Podsumowanie

- Zmienne globalne i lokalne
- Programowanie proceduralne - zmienne globalne z umiarem
- Deklaracja a definicja funkcji
- Funkcja przed użyciem musi być przynajmniej zadeklarowana
- `return` zwrócenie pojedynczej wartości
- Pytanie: czy można zwrócić z funkcji więcej wartości?
  Np. wyznaczanie miejsc zerowych paraboli.

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`