

Funkcje

Funkcje

Wartości w programie C

Funkcje

Wartości w programie C:

- (proste) **stałe**

Funkcje

Wartości w programie C:

- (proste) **stałe**, np. **-15**

Funkcje

Wartości w programie C:

- (proste) **stałe**, np. `-15`, `'+'`

Funkcje

Wartości w programie C:

- (proste) **stałe**, np. `-15`, `'+'`, `3.14159`

Funkcje

Wartości w programie C:

- (proste) **stałe**, np. `-15`, `'+'`, `3.14159`
- (proste) **zmienne**

Funkcje

Wartości w programie C:

- (proste) **stałe**, np. `-15`, `'+'`, `3.14159`
- (proste) **zmienne**, np.
`n` po deklaracji `int n;`

Funkcje

Wartości w programie C:

- (proste) **stałe**, np. `-15`, `'+'`, `3.14159`
- (proste) **zmienne**, np.

$\left. \begin{array}{l} n \\ c \end{array} \right\}$ po deklaracji $\left\{ \begin{array}{l} \text{int } n; \\ \text{char } c; \end{array} \right.$

Funkcje

Wartości w programie C:

- (proste) **stałe**, np. `-15`, `'+'`, `3.14159`
- (proste) **zmienne**, np.

<code>n</code>	}	po deklaracji	{	<code>int n;</code>
<code>c</code>				<code>char c;</code>
<code>x</code>				<code>double x;</code>

Funkcje

Wartości w programie C:

- (proste) **stałe**, np. `-15`, `'+'`, `3.14159`
- (proste) **zmienne**, np.

$\left. \begin{array}{l} n \\ c \\ x \end{array} \right\}$ po deklaracji $\left\{ \begin{array}{l} \text{int } n; \\ \text{char } c; \\ \text{double } x; \end{array} \right.$

- (złożone) wyrażenia zbudowane z użyciem **operatorów**

Funkcje

Wartości w programie C:

- (proste) **stałe**, np. `-15`, `'+'`, `3.14159`
- (proste) **zmienne**, np.

$\left. \begin{array}{l} n \\ c \\ x \end{array} \right\}$ po deklaracji $\left\{ \begin{array}{l} \text{int } n; \\ \text{char } c; \\ \text{double } x; \end{array} \right.$

- (złożone) wyrażenia zbudowane z użyciem **operatorów**, np.
`(6+(n-1))*(x%10+7)`

Funkcje

Wartości w programie C:

- (proste) **stałe**, np. `-15`, `'+'`, `3.14159`
- (proste) **zmienne**, np.

$\left. \begin{array}{l} n \\ c \\ x \end{array} \right\}$ po deklaracji $\left\{ \begin{array}{l} \text{int } n; \\ \text{char } c; \\ \text{double } x; \end{array} \right.$

- (złożone) wyrażenia zbudowane z użyciem **operatorów**, np.
`(6+(n-1))*(x%10+7)` , `x > 0 && x*x <= 2/x`

Funkcje

Wartości w programie C:

- (proste) **stałe**, np. `-15`, `'+'`, `3.14159`
- (proste) **zmienne**, np.

$\left. \begin{array}{l} n \\ c \\ x \end{array} \right\}$ po deklaracji $\left\{ \begin{array}{l} \text{int } n; \\ \text{char } c; \\ \text{double } x; \end{array} \right.$

- (złożone) wyrażenia zbudowane z użyciem **operatorów**, np.
`(6+(n-1))*(x%10+7)` , `x > 0 && x*x <= 2/x`
- (złożone) wyrażenia zbudowane z użyciem **funkcji**

Funkcje

Wartości w programie C:

- (proste) **stałe**, np. `-15`, `'+'`, `3.14159`
- (proste) **zmienne**, np.

$$\left. \begin{array}{l} n \\ c \\ x \end{array} \right\} \text{ po deklaracji } \left\{ \begin{array}{l} \text{int } n; \\ \text{char } c; \\ \text{double } x; \end{array} \right.$$

- (złożone) wyrażenia zbudowane z użyciem **operatorów**, np.
`(6+(n-1))*(x%10+7)` , `x > 0 && x*x <= 2/x`
- (złożone) wyrażenia zbudowane z użyciem **funkcji**, np.
`isspace(c)`

Funkcje

Wartości w programie C:

- (proste) **stałe**, np. `-15`, `'+'`, `3.14159`
- (proste) **zmienne**, np.

$$\left. \begin{array}{l} n \\ c \\ x \end{array} \right\} \text{ po deklaracji } \left\{ \begin{array}{l} \text{int } n; \\ \text{char } c; \\ \text{double } x; \end{array} \right.$$

- (złożone) wyrażenia zbudowane z użyciem **operatorów**, np.
`(6+(n-1))*(x%10+7)` , `x > 0 && x*x <= 2/x`
- (złożone) wyrażenia zbudowane z użyciem **funkcji**, np.
`isspace(c)` , `sin(x-M_PI/2)`

Funkcje

Wartości w programie C:

- (proste) **stałe**, np. `-15`, `'+'`, `3.14159`
- (proste) **zmienne**, np.

$$\left. \begin{array}{l} n \\ c \\ x \end{array} \right\} \text{ po deklaracji } \left\{ \begin{array}{l} \text{int } n; \\ \text{char } c; \\ \text{double } x; \end{array} \right.$$

- (złożone) wyrażenia zbudowane z użyciem **operatorów**, np.
`(6+(n-1))*(x%10+7)` , `x > 0 && x*x <= 2/x`
- (złożone) wyrażenia zbudowane z użyciem **funkcji**, np.
`isspace(c)` , `sin(x-M_PI/2)` , `floor(pow(x,-3)) == 2`

Funkcje

Wywołanie funkcji w programie:

`nazwa(argumenty)`

Funkcje

Wywołanie funkcji w programie:

`nazwa(argumenty)`

Funkcję poznajemy po nawiasach

Funkcje

Wywołanie funkcji w programie:

```
nazwa(argumenty)
```

Funkcję poznajemy po nawiasach — nawet jeśli funkcja nie używa żadnych argumentów, to *i tak* potrzebny jest nawias:

```
nazwa( )
```

Funkcje

Wywołanie funkcji w programie:

```
nazwa(argumenty)
```

Funkcję poznajemy po nawiasach — nawet jeśli funkcja nie używa żadnych argumentów, to *i tak* potrzebny jest nawias:

```
nazwa( )
```

Należy rozróżniać:

```
nazwa(...)
```

 — wywołanie funkcji

Funkcje

Wywołanie funkcji w programie:

```
nazwa(argumenty)
```

Funkcję poznajemy po nawiasach — nawet jeśli funkcja nie używa żadnych argumentów, to *i tak* potrzebny jest nawias:

```
nazwa( )
```

Należy rozróżniać:

`nazwa(...)` — wywołanie funkcji

`nazwa + (...)` — operacja na zmiennej i wyrażeniu w nawiasie

Funkcje

MOŻEMY TWORZYĆ WŁASNE FUNKCJE!

Funkcje

MOŻEMY TWORZYĆ WŁASNE FUNKCJE!

```
typ nazwa ( parametry formalne ) {  
    deklaracje i komendy  
    return wyrażenie;  
}
```

— definicja

Funkcje

MOŻEMY TWORZYĆ WŁASNE FUNKCJE!

```
typ nazwa ( parametry formalne ) {  
    deklaracje i komendy  
    return wyrażenie;  
}
```

obowiązuje
zgodność
typów

— definicja

Funkcje

MOŻEMY TWORZYĆ WŁASNE FUNKCJE!

```
typ nazwa ( parametry formalne ) {  
    deklaracje i komendy  
    return wyrażenie;  
}
```

obowiązuje
zgodność
typów

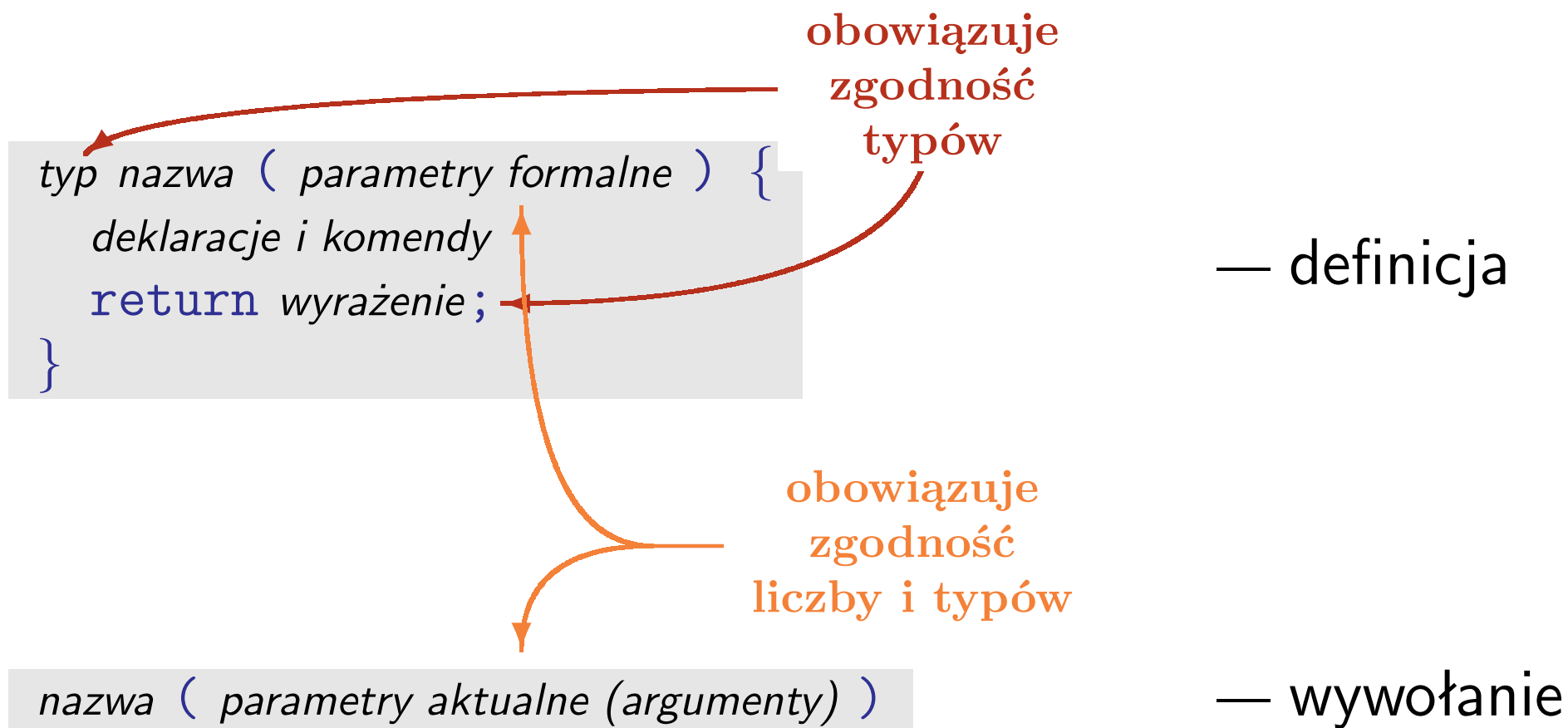
— definicja

```
nazwa ( parametry aktualne (argumenty) )
```

— wywołanie

Funkcje

MOŻEMY TWORZYĆ WŁASNE FUNKCJE!



Przykład: liczba podzielników:

```
#include<stdio.h>

int podzielniki(int k) {
    int lpodz=0, podz=1;
    while (podz <= k) {
        if (k%podz == 0) lpodz++;
        podz++;
    };
    return lpodz;
}
```

Przykład: liczba podzielników:

```
#include<stdio.h>

int podzielniki(int k) {
    int  lpodz=0, podz=1;
    while (podz <= k) {
        if (k%podz == 0)  lpodz++;
        podz++;
    };
    return lpodz;
}

int main() {
    int n;
    printf("\n n == "); scanf("%i", &n);
    printf(" %i podzielnikow\n\n", podzielniki(n));
    return 0;
}
```

Przykład: liczba podzielników:

```
#include<stdio.h>

int podzielniki(int k) {
    int  lpodz=0, podz=1;
    while (podz <= k) {
        if (k%podz == 0)  lpodz++;
        podz++;
    };
    return lpodz;
}

int main() {
    int n;
    printf("\n n == "); scanf("%i", &n);
    printf(" %i podzielnikow\n\n", podzielniki(n));
    return 0;
}
```

arg. formalny typu int

arg. aktualny typu int

Przykład: liczba podzielników:

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

```
int podzielniki(int k) {  
    int  lpodz=0, podz=1;  
    while (podz <= k) {  
        if (k%podz == 0)  lpodz++;  
        podz++;  
    };  
    return lpodz;  
}
```

arg. formalny typu int

```
int main() {  
    int n;  
    printf("\n n == "); scanf("%i", &n);  
    printf(" %i podzielnikow\n\n", podzielniki(n));  
    return 0;  
}
```

arg. aktualny typu int

miejsce na
wartość
całkowitą

wynik całkowity

Przykład: liczba podzielników:

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

```
int podzielniki(int k) {  
    int lpodz=0, podz=1;  
    while (podz <= k) {  
        if (k%podz == 0) lpodz++;  
        podz++;  
    };  
    return lpodz;  
}
```

arg. formalny typu int

n == 12

```
int main() {  
    int n;  
    printf("\n n == "); scanf("%i", &n);  
    printf(" %i podzielnikow\n\n", podzielniki(n));  
    return 0;  
}
```

arg. aktualny typu int

miejsce na
wartość
całkowitą

wynik całkowity

Przykład: liczba podzielników:

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

```
int podzielniki(int k) {  
    int lpodz=0, podz=1;  
    while (podz <= k) {  
        if (k%podz == 0) lpodz++;  
        podz++;  
    };  
    return lpodz;  
}
```

arg. formalny typu int

n == 12
6 podzielnikow

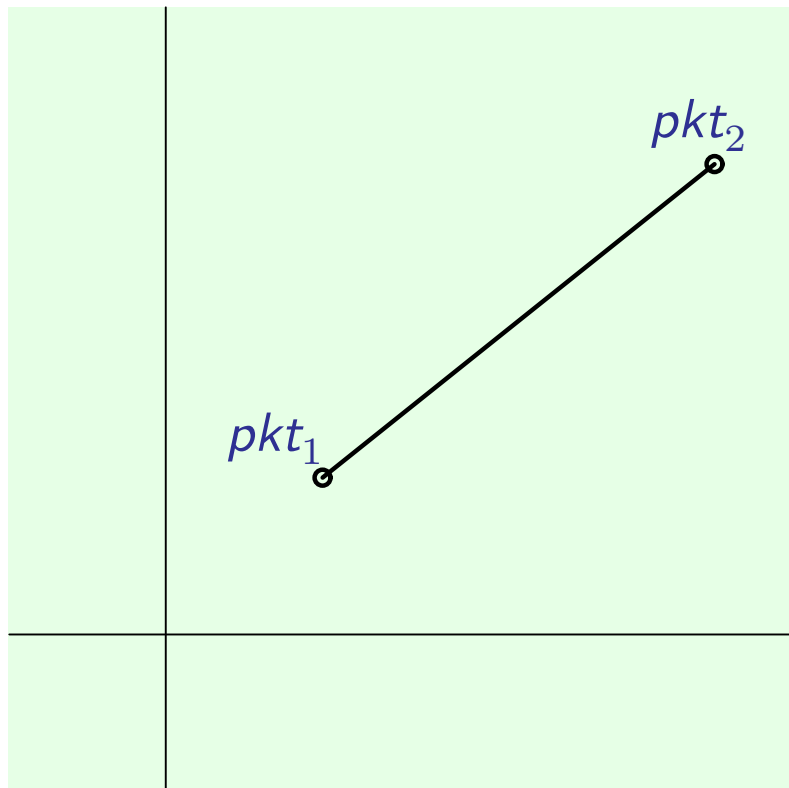
```
int main() {  
    int n;  
    printf("\n n == "); scanf("%i", &n);  
    printf(" %i podzielnikow\n\n", podzielniki(n));  
    return 0;  
}
```

arg. aktualny typu int

miejsce na
wartość
całkowitą

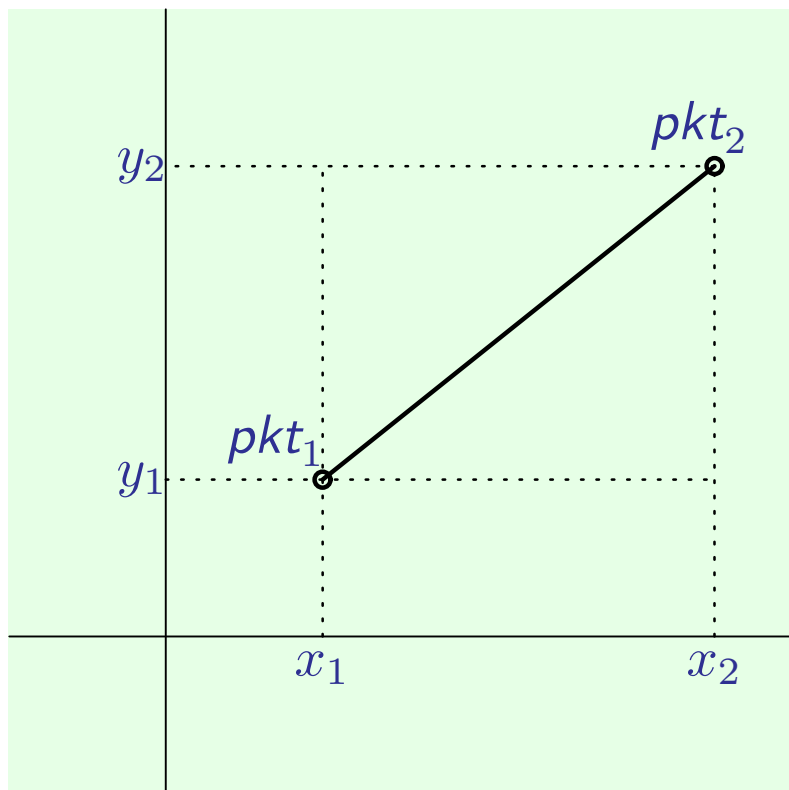
wynik całkowity

Przykład: odległość punktów



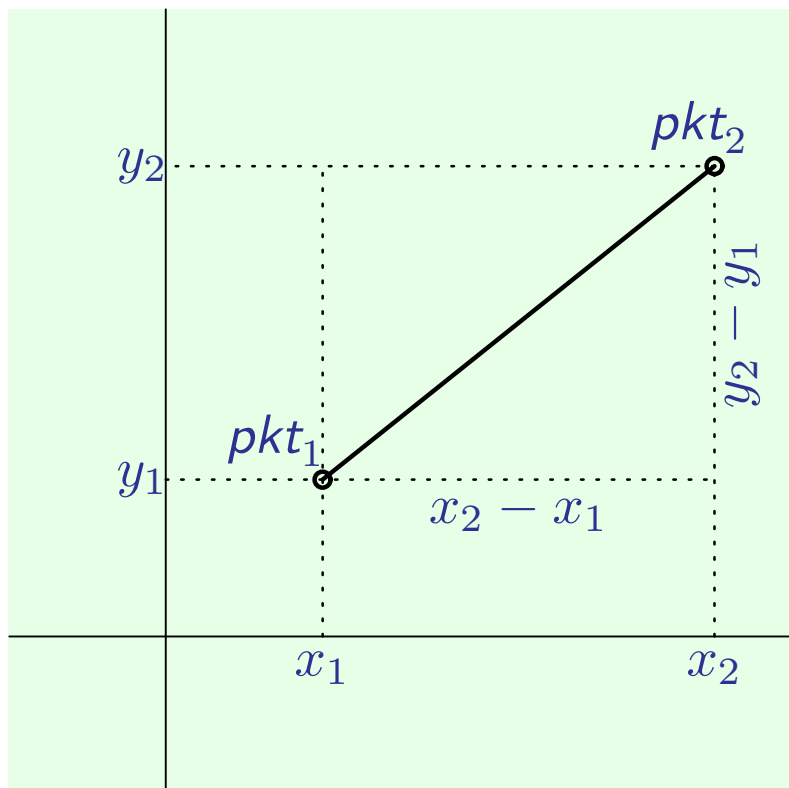
odległość pkt_1 do pkt_1 =

Przykład: odległość punktów



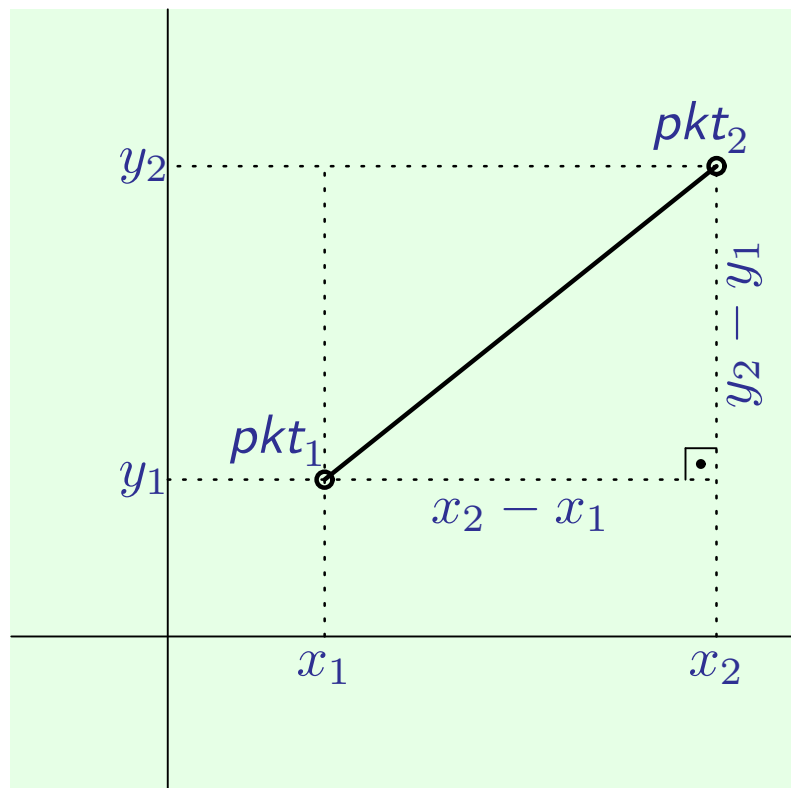
odległość pkt_1 do pkt_1 =

Przykład: odległość punktów



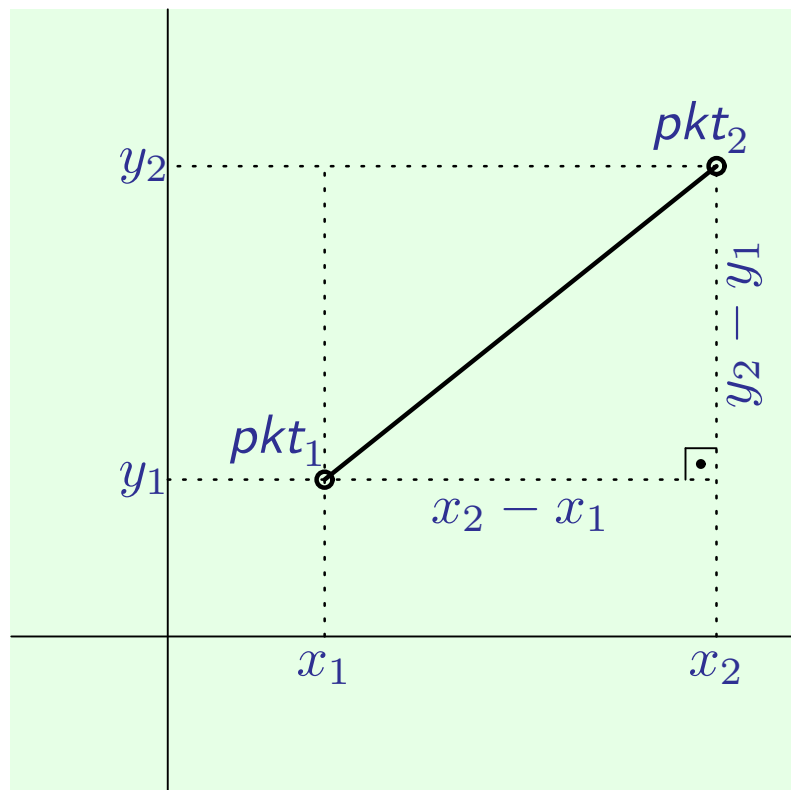
odległość pkt_1 do pkt_1 =

Przykład: odległość punktów



odległość pkt_1 do pkt_1 =

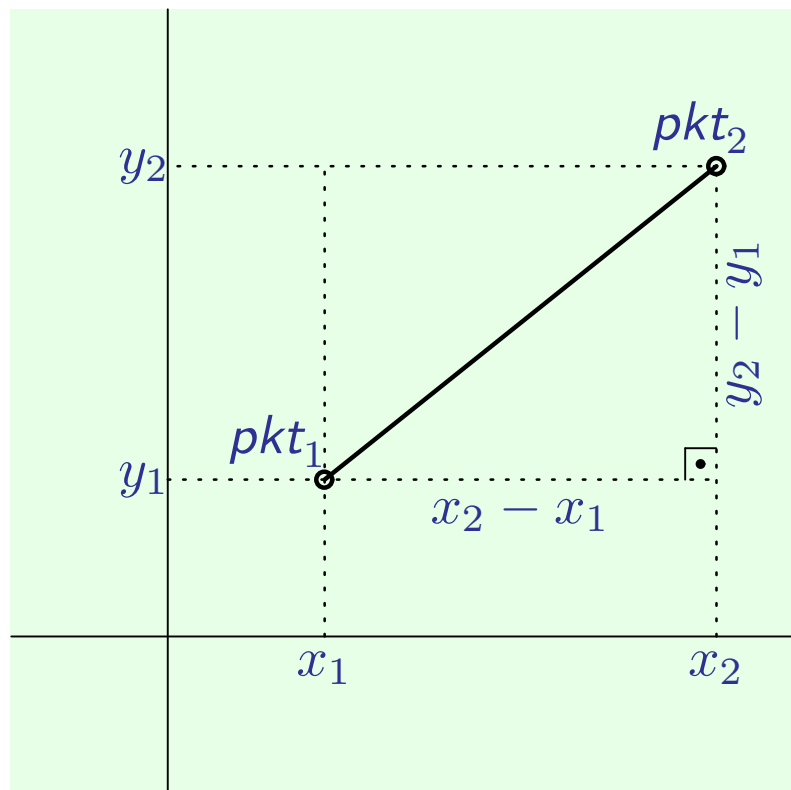
Przykład: odległość punktów



odległość pkt_1 do pkt_2 =

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Przykład: odległość punktów



odległość pkt_1 do pkt_1 =

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
#include<stdio.h>
#include<math.h>  ← żeby sqrt działał
```

```
double odleglosc(double x1, double y1, double x2, double y2) {
    double odl_x = x1-x2, odl_y = y1-y2;
    return sqrt(odl_x*odl_x + odl_y*odl_y);
}
```

Przykład: długość łamanej

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

double odleglosc(double x1, double y1, double x2, double y2) {
    double odl_x = x1-x2, odl_y = y1-y2;
    return sqrt(odl_x*odl_x + odl_y*odl_y);
}

int main() {

    return 0;
}
```

Przykład: długość łamanej

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

double odleglosc(double x1, double y1, double x2, double y2) {
    double odl_x = x1-x2, odl_y = y1-y2;
    return sqrt(odl_x*odl_x + odl_y*odl_y);
}

int main() {
    int n, i;
    printf("\n n == "); scanf("%i", &n);
    double x[n], y[n], dl_drogi = 0;

    return 0;
}
```

Przykład: długość łamanej

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

double odleglosc(double x1, double y1, double x2, double y2) {
    double odl_x = x1-x2, odl_y = y1-y2;
    return sqrt(odl_x*odl_x + odl_y*odl_y);
}

int main() {
    int n, i;
    printf("\n n == "); scanf("%i", &n);
    double x[n], y[n], dl_drogi = 0;
    for (i=0; i<n; i++) {
        printf(" (x[%i], y[%i]) == ", i, i);
        scanf("%lf", &x[i]); scanf("%lf", &y[i]);
        if (i>0)
            dl_drogi += odleglosc(x[i-1],y[i-1], x[i],y[i]);
    }

    return 0;
}
```

Przykład: długość łamanej

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

double odleglosc(double x1, double y1, double x2, double y2) {
    double odl_x = x1-x2, odl_y = y1-y2;
    return sqrt(odl_x*odl_x + odl_y*odl_y);
}

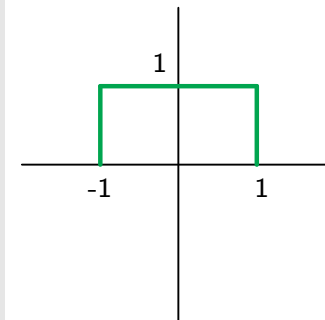
int main() {
    int n, i;
    printf("\n n == "); scanf("%i", &n);
    double x[n], y[n], dl_drogi = 0;
    for (i=0; i<n; i++) {
        printf(" (x[%i], y[%i]) == ", i, i);
        scanf("%lf", &x[i]); scanf("%lf", &y[i]);
        if (i>0)
            dl_drogi += odleglosc(x[i-1],y[i-1], x[i],y[i]);
    }
    printf("\n dlugosc drogi == %lf\n\n", dl_drogi);
    return 0;
}
```

Przykład: długość łamanej

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

double odleglosc(double x1, double y1, double x2, double y2) {
    double odl_x = x1-x2, odl_y = y1-y2;
    return sqrt(odl_x*odl_x + odl_y*odl_y);
}

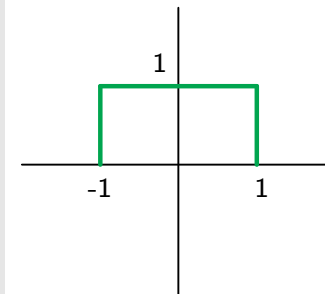
int main() {
    int n, i;
    printf("\n n == "); scanf("%i", &n);
    double x[n], y[n], dl_drogi = 0;
    for (i=0; i<n; i++) {
        printf(" (x[%i], y[%i]) == ", i, i);
        scanf("%lf", &x[i]); scanf("%lf", &y[i]);
        if (i>0)
            dl_drogi += odleglosc(x[i-1],y[i-1], x[i],y[i]);
    }
    printf("\n dlugosc drogi == %lf\n\n", dl_drogi);
    return 0;
}
```



Przykład: długość łamanej

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

double odleglosc(double x1, double y1, double x2, double y2) {
    double odl_x = x1-x2, odl_y = y1-y2;
    return sqrt(odl_x*odl_x + odl_y*odl_y);
}
```



```
int main() {
    int n, i;
    printf("\n n == "); scanf("%i", &n);
    double x[n], y[n], dl_drogi = 0;
    for (i=0; i<n; i++) {
        printf(" (x[%i], y[%i]) == ", i, i);
        scanf("%lf", &x[i]); scanf("%lf", &y[i]);
        if (i>0)
            dl_drogi += odleglosc(x[i-1],y[i-1], x[i],y[i]);
    }
    printf("\n dlugosc drogi == %lf\n\n", dl_drogi);
    return 0;
}
```

`n == 4`

`(x[0], y[0]) == -1 0`

`(x[1], y[1]) == -1 1`

`(x[2], y[2]) == 1 1`

`(x[3], y[3]) == 1 0`

`dlugosc drogi == 4.000000`

Przykład: długość łamanej

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

double odleglosc(double x1, double y1, double x2, double y2) {
    double odl_x = x1-x2, odl_y = y1-y2;
    return sqrt(odl_x*odl_x + odl_y*odl_y);
}

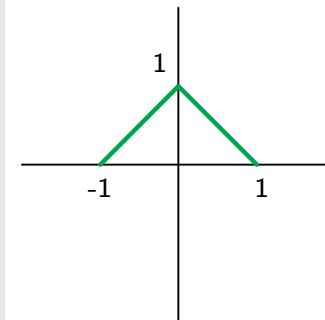
int main() {
    int n, i;
    printf("\n n == "); scanf("%i", &n);
    double x[n], y[n], dl_drogi = 0;
    for (i=0; i<n; i++) {
        printf(" (x[%i], y[%i]) == ", i, i);
        scanf("%lf", &x[i]); scanf("%lf", &y[i]);
        if (i>0)
            dl_drogi += odleglosc(x[i-1],y[i-1], x[i],y[i]);
    }
    printf("\n dlugosc drogi == %lf\n\n", dl_drogi);
    return 0;
}
```

Przykład: długość łamanej

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

double odleglosc(double x1, double y1, double x2, double y2) {
    double odl_x = x1-x2, odl_y = y1-y2;
    return sqrt(odl_x*odl_x + odl_y*odl_y);
}

int main() {
    int n, i;
    printf("\n n == "); scanf("%i", &n);
    double x[n], y[n], dl_drogi = 0;
    for (i=0; i<n; i++) {
        printf(" (x[%i], y[%i]) == ", i, i);
        scanf("%lf", &x[i]); scanf("%lf", &y[i]);
        if (i>0)
            dl_drogi += odleglosc(x[i-1],y[i-1], x[i],y[i]);
    }
    printf("\n dlugosc drogi == %lf\n\n", dl_drogi);
    return 0;
}
```

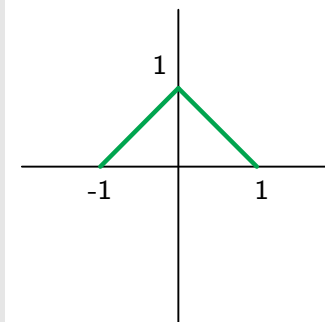


Przykład: długość łamanej

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

double odleglosc(double x1, double y1, double x2, double y2) {
    double odl_x = x1-x2, odl_y = y1-y2;
    return sqrt(odl_x*odl_x + odl_y*odl_y);
}
```

```
int main() {
    int n, i;
    printf("\n n == "); scanf("%i", &n);
    double x[n], y[n], dl_drogi = 0;
    for (i=0; i<n; i++) {
        printf(" (x[%i], y[%i]) == ", i, i);
        scanf("%lf", &x[i]); scanf("%lf", &y[i]);
        if (i>0)
            dl_drogi += odleglosc(x[i-1],y[i-1], x[i],y[i]);
    }
    printf("\n dlugosc drogi == %lf\n\n", dl_drogi);
    return 0;
}
```



`n == 3`

`(x[0], y[0]) == -1 0`

`(x[1], y[1]) == 0 1`

`(x[2], y[2]) == 1 0`

`dlugosc drogi == 2.828427`

Funkcje *bez* wyniku

Funkcje *bez wyniku*

Komendy w programie C:

- (proste) przypisania

Funkcje *bez* wyniku

Komendy w programie C:

- (proste) przypisania, np.

`n = n+1;` , `n += 1;` , `n++;` — równoważne

Funkcje *bez* wyniku

Komendy w programie C:

- (proste) przypisania, np.

`n = n+1;` , `n += 1;` , `n++;` — równoważne
`y = 2*pow(x+1,3)+1;`

Funkcje *bez* wyniku

Komendy w programie C:

- (proste) przypisania, np.

`n = n+1; , n += 1; , n++;` — równoważne
`y = 2*pow(x+1,3)+1;`

- (złożone) zbudowane przy pomocy

```
if ( ... ) ...else ...  
if ( ... ) ...
```

Funkcje *bez* wyniku

Komendy w programie C:

- (proste) przypisania, np.

`n = n+1;` , `n += 1;` , `n++;` — równoważne
`y = 2*pow(x+1,3)+1;`

- (złożone) zbudowane przy pomocy

`if ( ... ) ... else ...`
`if ( ... ) ...`
`while ( ... ) ...`

Funkcje *bez* wyniku

Komendy w programie C:

- (proste) przypisania, np.

`n = n+1; , n += 1; , n++;` — równoważne
`y = 2*pow(x+1,3)+1;`

- (złożone) zbudowane przy pomocy

```
if ( ... ) ...else ...  
if ( ... ) ...  
while ( ... ) ...  
do { ... } while ( ... );
```

Funkcje *bez* wyniku

Komendy w programie C:

- (proste) przypisania, np.

```
n = n+1; ,    n += 1; ,    n++;    — równoważne  
y = 2*pow(x+1,3)+1;
```

- (proste) wywołania funkcji bez wyniku, np.

```
printf("%lf", tab[1]);
```

- (złożone) zbudowane przy pomocy

```
if ( ... ) ...else ...  
if ( ... ) ...  
while ( ... ) ...  
do { ... } while ( ... );
```

Funkcje *bez* wyniku

WŁASNE FUNKCJE BEZ WYNIKU

Funkcje *bez* wyniku

WŁASNE FUNKCJE BEZ WYNIKU

```
void nazwa ( parametry formalne ) {  
    deklaracje i komendy  
}
```

— definicja

Funkcje *bez* wyniku

WŁASNE FUNKCJE BEZ WYNIKU

```
void nazwa ( parametry formalne ) {  
    deklaracje i komendy  
}
```

— definicja

```
nazwa ( parametry aktualne (argumenty) );
```

— wywołanie

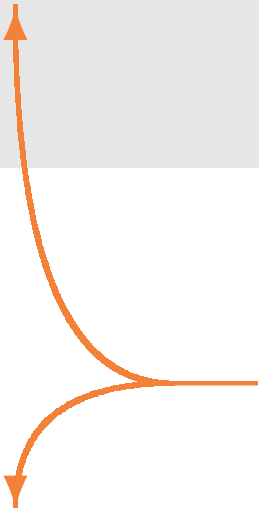
Funkcje *bez wyniku*

WŁASNE FUNKCJE BEZ WYNIKU

```
void nazwa ( parametry formalne ) {  
    deklaracje i komendy  
}
```

— definicja

obowiązuje
zgodność
liczby i typów



```
nazwa ( parametry aktualne (argumenty) );
```

— wywołanie

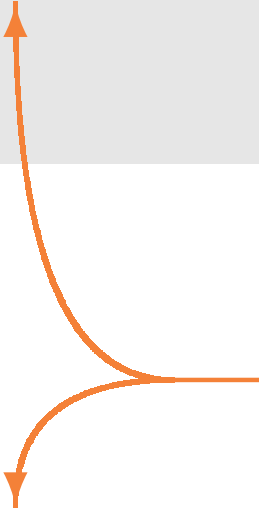
Funkcje *bez wyniku*

WŁASNE FUNKCJE BEZ WYNIKU

```
void nazwa ( parametry formalne ) {  
    deklaracje i komendy  
}
```

— definicja

obowiązuje
zgodność
liczby i typów



```
nazwa ( parametry aktualne (argumenty) );
```

— wywołanie

NIE MA:

- typu wynikowego funkcji (zamiast niego `void`)

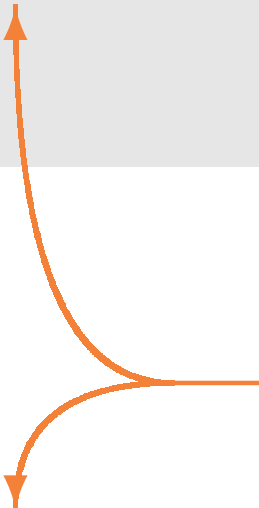
Funkcje *bez wyniku*

WŁASNE FUNKCJE BEZ WYNIKU

```
void nazwa ( parametry formalne ) {  
    deklaracje i komendy  
}
```

— definicja

obowiązuje
zgodność
liczby i typów



```
nazwa ( parametry aktualne (argumenty) );
```

— wywołanie

NIE MA:

- typu wynikowego funkcji (zamiast niego `void`),
- komendy `return`.

Przykład: słowa w ramkach

```
+-----+   +---+   +-----+
| słowa |   | w |   | ramkach |
+-----+   +---+   +-----+
```

Przykład: słowa w ramkach

```
+-----+   +---+   +-----+  
|  słowa  |   | w  |   | ramkach |  
+-----+   +---+   +-----+
```

```
#include<stdio.h>  
  
int main() {  
    printf("\n"); pozioma();  
    printf("  | słowa |    | w |    | ramkach |\n");  
    pozioma(); printf("\n");  
}
```

Przykład: słowa w ramkach

+-----+	+---+	+-----+
słowa	w	ramach
+-----+	+---+	+-----+

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

```
int main() {  
    printf("\n"); pozioma();  
    printf(" | słowa |   | w |   | ramach |\n");  
    pozioma(); printf("\n");  
}
```

Przykład: słowa w ramkach

```
+-----+   +---+   +-----+  
| słowa |   | w |   | ramkach |  
+-----+   +---+   +-----+
```

```
#include<stdio.h>  
  
int main() {  
    printf("\n"); pozioma();  
    printf(" | słowa |   | w |   | ramkach |\n");  
    pozioma(); printf("\n");  
}
```

Przykład: słowa w ramkach

+-----+	+---+	+-----+
słowa	w	ramach
+-----+	+---+	+-----+

```
#include<stdio.h>

int main() {
    printf("\n"); pozioma();
    printf(" | słowa |   | w |   | ramach |\n");
    pozioma(); printf("\n");
}
```

Przykład: słowa w ramkach

```
+-----+   +---+   +-----+  
| słowa |   | w |   | ramkach |  
+-----+   +---+   +-----+
```

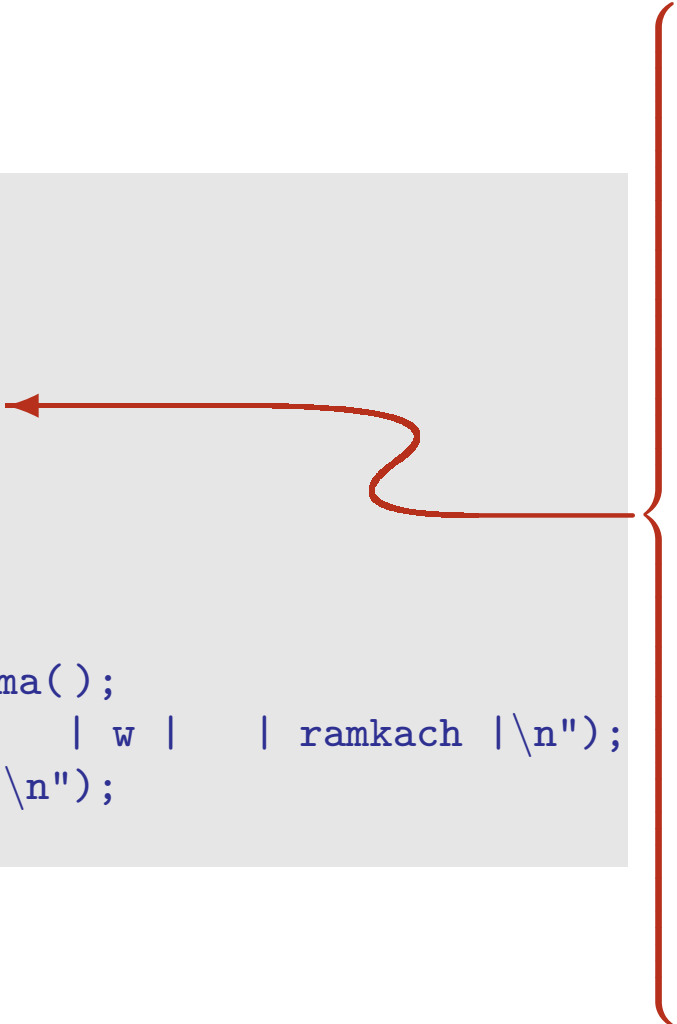
```
#include<stdio.h>  
  
int main() {  
    printf("\n"); pozioma();  
    printf("  | słowa |    | w |    | ramkach |\n");  
    pozioma(); printf("\n");  
}
```

Przykład: słowa w ramkach

```
+-----+   +---+   +-----+  
| słowa |   | w |   | ramach |  
+-----+   +---+   +-----+
```

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

```
int main() {  
    printf("\n"); pozioma();  
    printf(" | słowa |   | w |   | ramach |\n");  
    pozioma(); printf("\n");  
}
```



Przykład: słowa w ramkach

```
+-----+   +---+   +-----+
| słowa |   | w |   | ramach |
+-----+   +---+   +-----+
```

```
#include<stdio.h>

int main() {
    printf("\n"); pozioma();
    printf(" | słowa |   | w |   | ramach |\n");
    pozioma(); printf("\n");
}
```

```
void pozioma() {
    spacje(2); kreski(7);
    spacje(3); kreski(3);
    spacje(3); kreski(9);
    printf("\n");
}
```

Przykład: słowa w ramkach

```
+-----+   +---+   +-----+
| słowa |   | w |   | ramach |
+-----+   +---+   +-----+
```

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

```
int main() {
    printf("\n"); pozioma();
    printf(" | słowa |   | w |   | ramach |\n");
    pozioma(); printf("\n");
}
```

```
void spacje(int n) {
    int i;
    for (i=0; i<n; i++) printf(" ");
}
```

```
void pozioma() {
    spacje(2); kreski(7);
    spacje(3); kreski(3);
    spacje(3); kreski(9);
    printf("\n");
}
```

Przykład: słowa w ramkach

```
+-----+   +---+   +-----+
|  słowa  |   | w |   | ramkach |
+-----+   +---+   +-----+
```

```
#include<stdio.h>

int main() {
    printf("\n"); pozioma();
    printf(" | słowa |   | w |   | ramkach |\n");
    pozioma(); printf("\n");
}
```

```
void spacje(int n) {
    int i;
    for (i=0; i<n; i++) printf(" ");
}

void kreski(int n) {
    int i;
    printf("+");
    for (i=0; i<n; i++) printf("-");
    printf("+");
}

void pozioma() {
    spacje(2); kreski(7);
    spacje(3); kreski(3);
    spacje(3); kreski(9);
    printf("\n");
}
```

Typy wyniku i parametrów funkcji

Typy wyniku i parametrów funkcji

Wynikiem funkcji nie-`void` może być:

- liczba całkowita

Typy wyniku i parametrów funkcji

Wynikiem funkcji nie-`void` może być:

- liczba całkowita,
- liczba rzeczywista

Typy wyniku i parametrów funkcji

Wynikiem funkcji nie-`void` może być:

- liczba całkowita,
- liczba rzeczywista,
- znak (`char`)

Typy wyniku i parametrów funkcji

Wynikiem funkcji nie-`void` może być:

- liczba całkowita,
- liczba rzeczywista,
- znak (`char`),
- wskaźnik (*dotąd nie omawialiśmy*).

Typy wyniku i parametrów funkcji

Wynikiem funkcji nie-void może być:

- liczba całkowita,
- liczba rzeczywista,
- znak (`char`),
- wskaźnik (*dotąd nie omawialiśmy*).

Argumentem funkcji może być:

- liczba całkowita,
- liczba rzeczywista,
- znak (`char`),
- wskaźnik (*dotąd nie omawialiśmy*)

Typy wyniku i parametrów funkcji

Wynikiem funkcji nie-void może być:

- liczba całkowita,
- liczba rzeczywista,
- znak (`char`),
- wskaźnik (*dotąd nie omawialiśmy*).

Argumentem funkcji może być:

- liczba całkowita,
- liczba rzeczywista,
- znak (`char`),
- wskaźnik (*dotąd nie omawialiśmy*),
- tablica (dowolnie wielowymiarowa, dowolnego typu elementów)

Typy wyniku i parametrów funkcji

Wynikiem funkcji nie-void może być:

- liczba całkowita,
- liczba rzeczywista,
- znak (`char`),
- wskaźnik (*dotąd nie omawialiśmy*).

Argumentem funkcji może być:

- liczba całkowita,
- liczba rzeczywista,
- znak (`char`),
- wskaźnik (*dotąd nie omawialiśmy*),
- tablica (dowolnie wielowymiarowa, dowolnego typu elementów),
- struktura (*dotąd nie omawialiśmy*)

Typy wyniku i parametrów funkcji

Wynikiem funkcji nie-void może być:

- liczba całkowita,
- liczba rzeczywista,
- znak (`char`),
- wskaźnik (*dotąd nie omawialiśmy*).

Argumentem funkcji może być:

- liczba całkowita,
- liczba rzeczywista,
- znak (`char`),
- wskaźnik (*dotąd nie omawialiśmy*),
- tablica (dowolnie wielowymiarowa, dowolnego typu elementów),
- struktura (*dotąd nie omawialiśmy*),
- plik (*dotąd nie omawialiśmy*).

Typy wyniku i parametrów funkcji

Wynikiem funkcji nie-`void` może być:

- liczba całkowita,
- liczba rzeczywista,
- znak (`char`),
- wskaźnik (*dotąd nie omawialiśmy*).

Argumentem funkcji może być:

- liczba całkowita,
- liczba rzeczywista,
- znak (`char`),
- wskaźnik (*dotąd nie omawialiśmy*),
- tablica (dowolnie wielowymiarowa, dowolnego typu elementów),
- struktura (*dotąd nie omawialiśmy*),
- plik (*dotąd nie omawialiśmy*).

W szczególności tablica **nie może** być wynikiem funkcji:

```
int[100] funkcja(int n) { ... }
```

Typy wyniku i parametrów funkcji

Wynikiem funkcji nie-`void` może być:

- liczba całkowita,
- liczba rzeczywista,
- znak (`char`),
- wskaźnik (*dotąd nie omawialiśmy*).

Argumentem funkcji może być:

- liczba całkowita,
- liczba rzeczywista,
- znak (`char`),
- wskaźnik (*dotąd nie omawialiśmy*),
- tablica (dowolnie wielowymiarowa, dowolnego typu elementów),
- struktura (*dotąd nie omawialiśmy*),
- plik (*dotąd nie omawialiśmy*).

W szczególności tablica **nie może** być wynikiem funkcji:

~~`int[100] funkcja(int n) { ... }`~~

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera
po wspólnym prefiksie

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park

partia

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

par**k**

part**i**a

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park
partia



Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park
partia



kreska
krata

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park
partia



kreska
krata

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park



partia

kreska

krata



Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park
partia



kreska
krata



koszyk
kos

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park
partia



kreska
krata



koszyk
kos

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park
partia



kreska
krata



koszyk
kos



Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park
partia



kreska
krata



koszyk
kos



```
int poprzedza(int a_dlug, char a[], int b_dlug, char b[])
```

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park
partia



kreska
krata



koszyk
kos



```
int poprzedza(int a_dlug, char a[], int b_dlug, char b[])
```

a poprzedza b: wynik -1
a równe b: wynik 0
b poprzedza a: wynik 1

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park
partia



kreska
krata



koszyk
kos



```
int poprzedza(int a_dlug, char a[], int b_dlug, char b[]) {  
    int i=0;  
    // pomijanie wspolnego prefiksu:  
    while (i<a_dlug && i<b_dlug && a[i]==b[i]) i++;
```

a poprzedza b: wynik -1
a równe b: wynik 0
b poprzedza a: wynik 1

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park
partia



kreska
krata



koszyk
kos



```
int poprzedza(int a_dlug, char a[], int b_dlug, char b[]) {  
    int i=0;  
    // pomijanie wspolnego prefiksu:  
    while (i<a_dlug && i<b_dlug && a[i]==b[i]) i++;  
    if (i == a_dlug) // a sie skonczylo
```

a poprzedza b: wynik -1
a równe b: wynik 0
b poprzedza a: wynik 1

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park
partia



kreska
krata



koszyk
kos



```
int poprzedza(int a_dlug, char a[], int b_dlug, char b[]) {
    int i=0;
    // pomijanie wspolnego prefiksu:
    while (i<a_dlug && i<b_dlug && a[i]==b[i]) i++;
    if (i == a_dlug) // a sie skonczylo
        if (i == b_dlug) // oba skonczyly sie jednocześnie
            return 0;
```

a poprzedza b: wynik -1
a równe b: wynik 0
b poprzedza a: wynik 1

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park
partia



kreska
krata



koszyk
kos



```
int poprzedza(int a_dlug, char a[], int b_dlug, char b[]) {  
    int i=0;  
    // pomijanie wspolnego prefiksu:  
    while (i<a_dlug && i<b_dlug && a[i]==b[i]) i++;  
    if (i == a_dlug) // a sie skonczylo  
        if (i == b_dlug) // oba skonczyly sie jednoczesnie  
            return 0;  
    else // a sie skonczylo, b nie  
        return -1;  
}
```

a poprzedza b: wynik -1
a równe b: wynik 0
b poprzedza a: wynik 1

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park
partia



kreska
krata



koszyk
kos



```
int poprzedza(int a_dlug, char a[], int b_dlug, char b[]) {  
    int i=0;  
    // pomijanie wspolnego prefiksu:  
    while (i<a_dlug && i<b_dlug && a[i]==b[i]) i++;  
    if (i == a_dlug) // a sie skonczylo  
        if (i == b_dlug) // oba skonczyly sie jednocześnie  
            return 0;  
        else // a sie skonczylo, b nie  
            return -1;  
    else // a sie nie skonczylo
```

a poprzedza b: wynik -1
a równe b: wynik 0
b poprzedza a: wynik 1

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park
partia



kreska
krata



koszyk
kos



```
int poprzedza(int a_dlug, char a[], int b_dlug, char b[]) {
    int i=0;
    // pomijanie wspolnego prefiksu:
    while (i<a_dlug && i<b_dlug && a[i]==b[i]) i++;
    if (i == a_dlug) // a sie skonczylo
        if (i == b_dlug) // oba skonczyly sie jednoczesnie
            return 0;
        else // a sie skonczylo, b nie
            return -1;
    else // a sie nie skonczylo
        if (i == b_dlug) // b sie skonczylo, a nie
            return 1;
```

a poprzedza b: wynik -1
a równe b: wynik 0
b poprzedza a: wynik 1

Przykład: kolejność alfabetyczna

O kolejności alfabetycznej słów decyduje pierwsza litera **po wspólnym prefiksie**, *nic* poprzedza wszystkie litery; np.

park
partia



kreska
krata



koszyk
kos



```
int poprzedza(int a_dlug, char a[], int b_dlug, char b[]) {
    int i=0;
    // pomijanie wspolnego prefiksu:
    while (i<a_dlug && i<b_dlug && a[i]==b[i]) i++;
    if (i == a_dlug) // a sie skonczylo
        if (i == b_dlug) // oba skonczyly sie jednoczesnie
            return 0;
        else // a sie skonczylo, b nie
            return -1;
    else // a sie nie skonczylo
        if (i == b_dlug) // b sie skonczylo, a nie
            return 1;
        else // zadne sie nie skonczylo, litera i rozna
            if (a[i] < b[i]) return -1;
            else return 1;
}
```

a poprzedza b: wynik -1
a równe b: wynik 0
b poprzedza a: wynik 1