



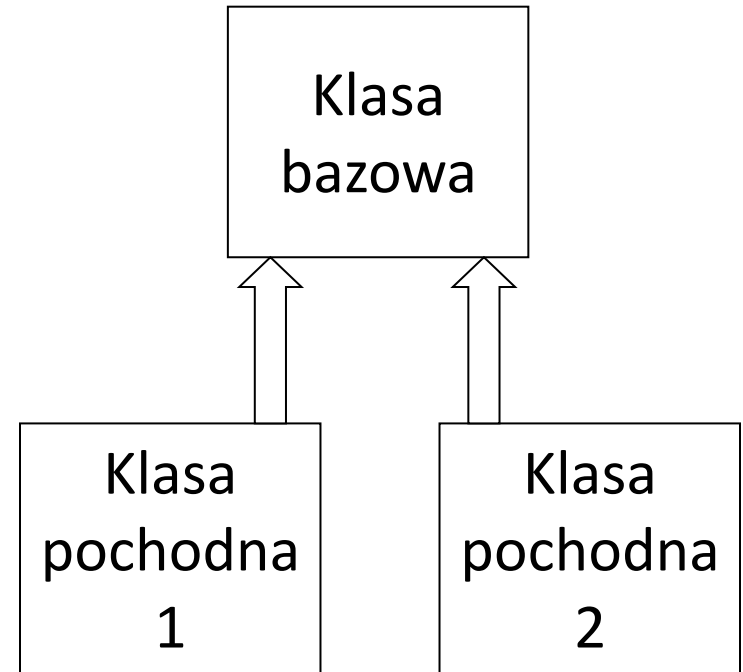
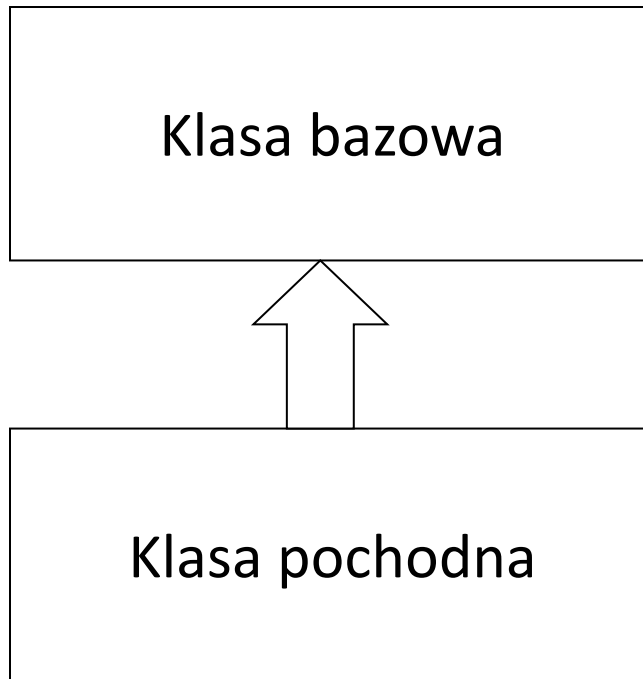
# Podstawy Programowania

dr inż. Tomasz Marciniak

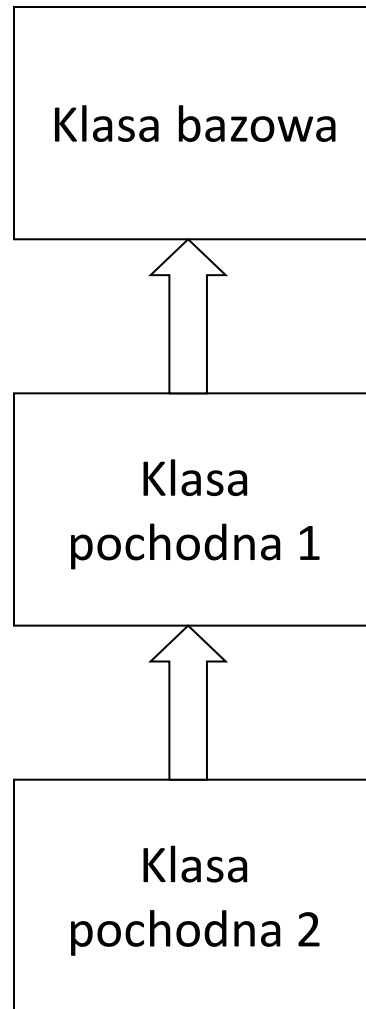
# Plan wykładu

- Dziedziczenie
- Funkcje zaprzyjaźnione
- Przeciążanie operatorów
- Typ wyliczeniowy

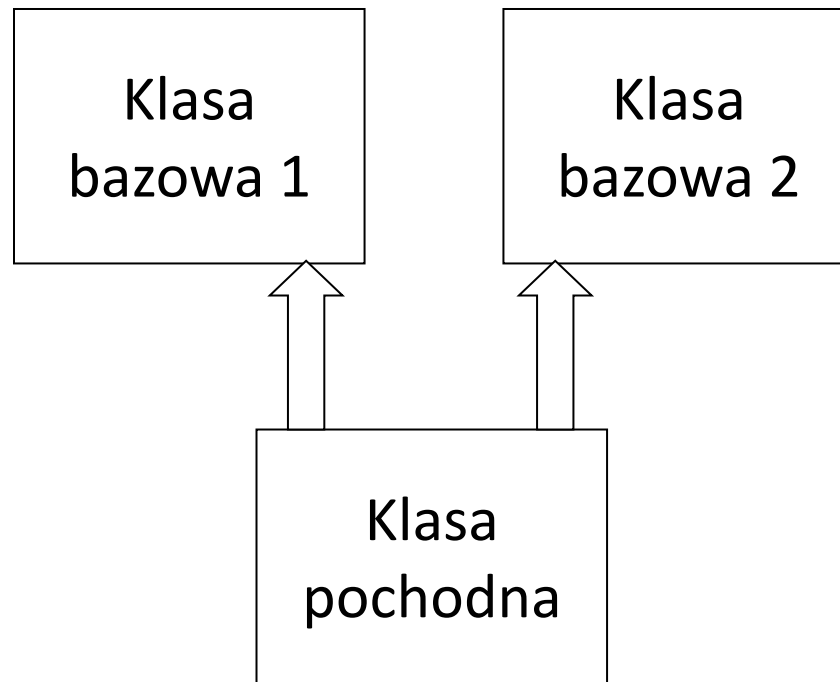
# Dziedziczenie proste



# Dziedziczenie proste kaskadowe



# Dziedziczenie mnogie



# Notacja UML (Unified Modeling Language)

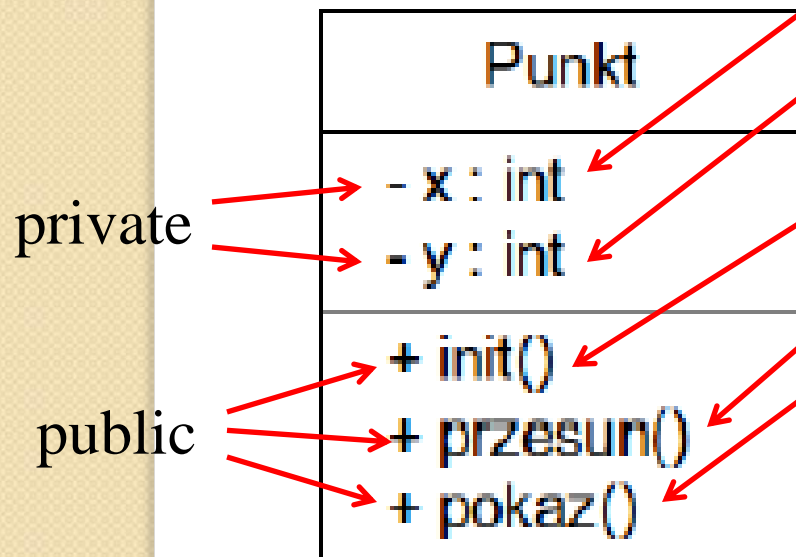
Nazwa klasy
Dane
Metody

Punkt
- x : int - y : int
+ init() + przesun() + pokaz()

```
class Punkt{  
    private:  
    int x;  
    int y;  
    public:  
    void init(int,int) ;  
    void przesun(int,int);  
    void pokaz() ;  
};
```

# Notacja UML (Unified Modeling Language)

Nazwa klasy
Dane
Metody



```
class Punkt{  
    private:  
        int x;  
        int y;  
    public:  
        void init(int,int) ;  
        void przesun(int,int);  
        void pokaz() ;  
};
```

# Przykład – klasa punkt

```
class punkt
{ double x , y ;
  public :
  void init(double xx=0.0,double yy=0.0)
    { x = xx;
      y = yy ;
    }
  void pokaz ( )
    { cout << "wspol rzedne : " << x << " " <<
      y << endl ;
    }
  double wsX( ) { return x ; }
  double wsY( ) { return y ; }
};
```



# Przykład – klasa punktB

```
class punktB : public punkt
{
    public :
    double promien ( )
        { return sqrt ( wsX( ) wsX( ) + wsY( ) wsY( ) ) ; }
};
```

**Dziedziczymy z klasy punkt**

```
int main ( )
{
    punktB a ;
    a . i n i t ( 3 , 4 ) ;
    a . pokaz ( ) ;
    cout << "promien      : " << a . promien ( ) ;
    getch ( ) ;
    return 0 ;
}
```

# Przykład – klasa punktB

```
class punktB : public punkt
{
    public :
    double promien ( )
        { return sqrt ( wsX( ) wsX( ) + wsY( ) wsY( ) ) ; }
};
```

**Dodajemy metodę  
obliczającą promień**



```
int main ( )
{
    punktB a ;
    a . i n i t ( 3 , 4 ) ;
    a . pokaz ( ) ;
    cout << "promien      : " << a . promien ( ) ;
    getch ( ) ;
    return 0 ;
}
```

# Przykład – klasa punktB

```
class punktB : public punkt
{
    public :
    double promien ( )
        { return sqrt ( wsX( ) wsX( ) + wsY( ) wsY( ) ) ; }
};
```

**Definiujemy obiekt a**


```
int main ( )
{
    punktB a ;
    a . i n i t ( 3 , 4 ) ;
    a . pokaz ( ) ;
    cout << "promien      : " << a . promien ( ) ;
    getch ( ) ;
    return 0 ;
}
```

# Przykład – klasa punktB

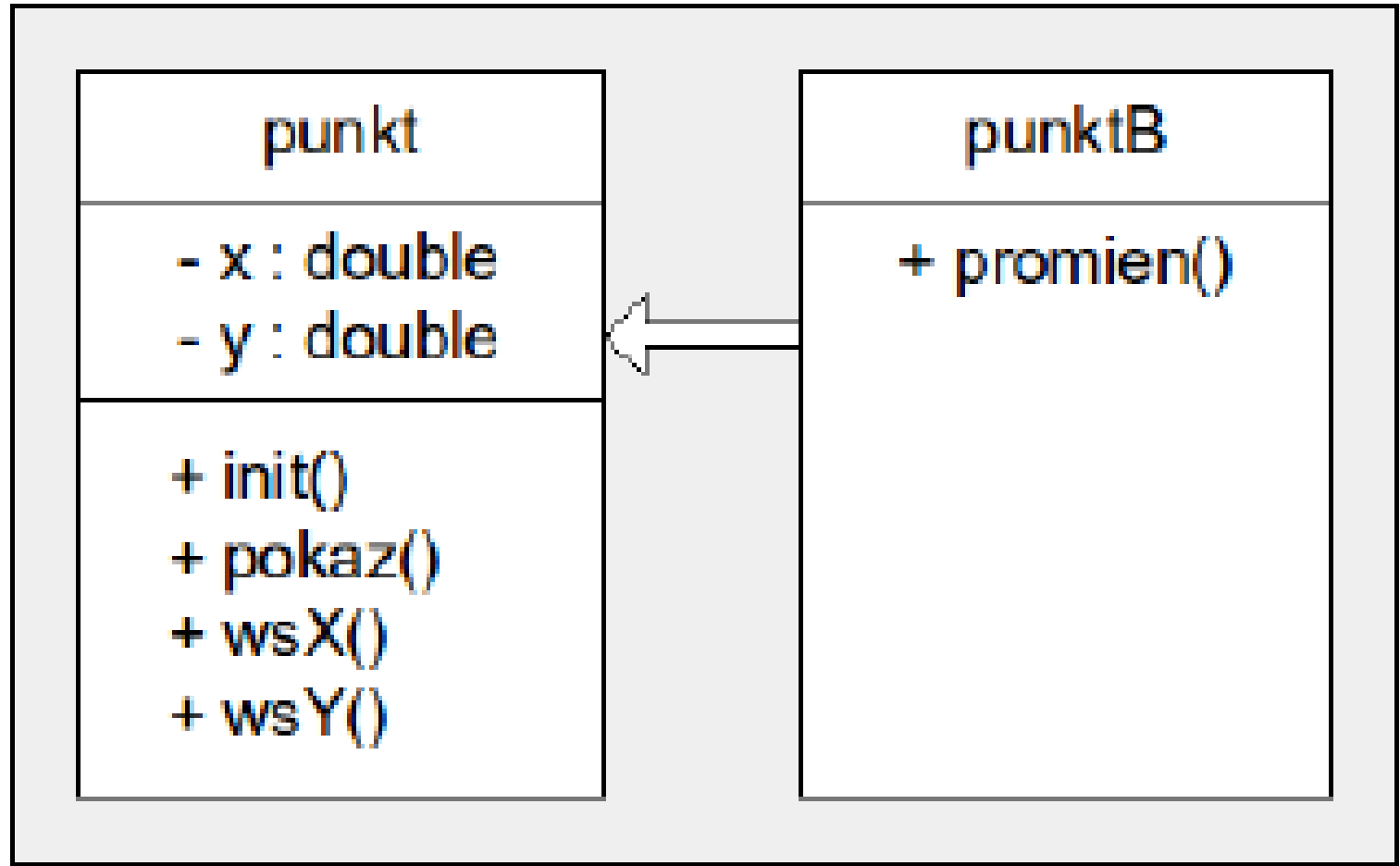
```
class punktB : public punkt
{
    public :
    double promien ( )
        { return sqrt (wsX() * wsX() + wsY() * wsY()) ;}
};
```

```
int main ( )
{
    punktB a ;
    a.init ( 3 , 4 ) ;
    a.pokaz ( ) ;
    cout << "promien      : " << a . promien ( ) ;
    getch ( ) ;
    return 0 ;
}
```

**Korzystamy z metody  
klasy bazowej init i  
pokaz**



# Diagram klas w UML



# Konstruktor klasy pochodnej i bazowej

- NIE DZIEDZICZYMY KONSTRUKTORÓW i DESTRUKTORÓW !!!
- Jeśli nie zdecydujemy inaczej to podczas tworzenia obiektu pochodnego zostanie wywołany domyślny konstruktor obiektu bazowego.

# Konstruktor klasy pochodnej i bazowej

Jak to zadziała ?

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class kl_A{
5      public:
6      int x,y;
7      kl_A(){x=2;y=4;cout << "domyslny klasy A"<<endl;}
8  };
9  class kl_B : public kl_A{
10     public:
11     int z;
12 };
13 int main(){
14     kl_B B;
15     cout << B.x << "    " << B.y << "    " << B.z ;
16     getch();
17 }
```

# Konstruktor klasy pochodnej i bazowej

Klasa A ma konstruktor domyślny

Klasa B nie ma konstruktora

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class kl_A{
5      public:
6      int x,y;
7      kl_A(){x=2;y=4;cout << "domyślny klasy A"<<endl;}
8  };
9  class kl_B : public kl_A{
10     public:
11     int z;
12 };
13 int main(){
14     kl_B B;
15     cout << B.x << "    " << B.y << "    " << B.z ;
16     getch();
17 }
```

```
domyslny klasy A
2    4    0
```



# Konstruktor klasy pochodnej i bazowej

Jak to zadziała ?

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class kl_A{
5      public:
6      int x,y;
7      kl_A(){ x=2;y=4;cout << "domyslny klasy A"<<endl;}
8  };
9  class kl_B : public kl_A{
10     public:
11     int z;
12     kl_B(int a, int b, int c){
13         z=c;cout << "konstruktor klasy B"<<endl;
14     }
15 };
16 int main(){
17     kl_B B(11,12,13);
18     cout << B.x << "    " << B.y << "    " << B.z ;
19     getch();
20 }
```

# Konstruktor klasy pochodnej i bazowej

Klasa A ma konstruktor domyślny

Klasa B ma konstruktor

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class kl_A{
5      public:
6      int x,y;
7      kl_A(){ x=2;y=4;cout << "domyślny klasy A"; }
8  };
9  class kl_B : public kl_A{
10     public:
11     int z;
12     kl_B(int a, int b, int c){
13         z=c;cout << "konstruktor klasy B"<<endl;
14     }
15 };
16 int main(){
17     kl_B B(11,12,13);
18     cout << B.x << " " << B.y << " " << B.z ;
19     getch();
20 }
```

```
domyslny klasy A
konstruktor klasy B
2   4   13
```

# Konstruktor klasy pochodnej i bazowej

Jak to zadziała ?

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class kl_A{
5      public:
6      int x,y;
7      kl_A(){ x=2;y=4;cout<<"domyslny klasy A"<<endl;}
8      kl_A(int a, int b){x=a,y=b;cout<<"konstruktor klasy A"<<endl;}
9  };
10 class kl_B : public kl_A{
11     public:
12     int z;
13     kl_B(int a,int b,int c){z=c;cout<<"konstruktor klasy B"<<endl;}
14 };
15 int main(){
16     kl_B B(11,12,13);
17     cout << B.x << "    " << B.y << "    " << B.z ;
18     getch();
19 }
```

# Konstruktor klasy pochodnej i bazowej

Klasa A ma 2 konstruktory domyślny i z dwoma parametrami

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class kl_A{
5      public:
6      int x,y;
7      kl_A(){ x=2;y=4;cout<<"domyślny klasy A"<<endl;}
8      kl_A(int a, int b){x=a,y=b;cout<<"konstruktor klasy A"<<endl;}
9  };
10 class kl_B : public kl_A{
11     public:
12     int z;
13     kl_B(int a,int b,int c){z=c;cout<<"konstruktor klasy B"<<endl;}
14 };
15 int main(){
16     kl_B B(11,12,13);
17     cout << B.x << "    " << B.y << "    " << B.z ;
18     getch();
19 }
```

Klasa B ma konstruktor

```
domyślny klasy A
konstruktor klasy B
2    4    13
```

# Konstruktor klasy pochodnej i bazowej

Jak to zadziała ?

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class kl_A{
5      public:
6      int x,y;
7      kl_A(){ x=2;y=4;cout<<"domyslny klasy A"<<endl;}
8      kl_A(int a, int b){x=a,y=b;cout<<"konstruktor klasy A"<<endl;}
9  };
10 class kl_B : public kl_A{
11     public:
12     int z;
13     kl_B(int a,int b,int c):kl_A(a,b)
14     {z=c;cout<<"konstruktor klasy B"<<endl;}
15 };
16 int main(){
17     kl_B B(11,12,13);
18     cout << B.x << "    " << B.y << "    " << B.z ;
19     getch();
20 }
```



# Konstruktor klasy pochodnej i bazowej

Klasa A ma 2 konstruktory

Klasa B ma konstruktor i wywołuje konstruktor klasy bazowej

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class kl_A{
5      public:
6      int x,y;
7      kl_A(){ x=2;y=4;cout<<"domyslny klasy A"<<endl;}
8      kl_A(int a, int b){x=a;y=b;cout<<"konstruktor klasy A"<<endl;}
9  };
10 class kl_B : public kl_A{
11     public:
12     int z;
13     kl_B(int a,int b,int c):kl_A(a,b)
14     {z=c;cout<<"konstruktor klasy B"<<endl;}
15 };
16 int main(){
17     kl_B B(11,12,13);
18     cout << B.x << "    " << B.y << "    " << B.z ;
19     getch();
20 }
```

```
konstruktor klasy A
konstruktor klasy B
11    12    13
```

# Dziedziczenie wielokrotne

```
class pochodna_1 : public bazowa_1 , public  
    bazowa_2  
{  
    // cialoklasy pochodna  
};
```

# Dziedziczenie wielokrotne

Jak to zadziała ?

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class kl_A{
5      public:
6      int x,y;
7      kl_A(){ x=2;y=4;cout<<"domyslny klasy A"<<endl;}
8  };
9  class kl_B{
10     public:
11     int w,h;
12     kl_B(){ w=22;h=24;cout<<"domyslny klasy B"<<endl;}
13 };
14 class kl_C : public kl_A , public kl_B{
15     public:
16     int z;
17     kl_C(int a,int b,int c)
18     {z=c;cout<<"konstruktor klasy C"<<endl;}
19 };
20 int main(){
21     kl_C C(11,12,13);
22     cout<<C.x<<"    "<<C.y<<"    "<<C.z<<"    "<<C.w<<"    "<<C.h;
23     getch();
24 }
```



# Dziedziczenie wielokrotne

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class kl_A{
5      public:
6      int x,y;
7      kl_A(){ x=2;y=4;cout<<"domyslny klasy A"<<endl;}
8  };
9  class kl_B{
10     public:
11     int w,h;
12     kl_B(){ w=22;h=24;cout<<"domyslny klasy B"<<endl;}
13 };
14 class kl_C : public kl_A , public kl_B{
15     public:
16     int z;
17     kl_C(int a,int b,int c)
18     {z=c;cout<<"konstruktor klasy C"<<endl;}
19 };
20 int main(){
21     kl_C C(11,12,13);
22     cout<<C.x<<" " <<C.y<<" " <<C.z<<" " <<C.w<<" " <<C.h;
23     getch();
24 }
```

```
domyslny klasy A
domyslny klasy B
konstruktor klasy C
2  4  13  22  24
```

# Funkcje zaprzyjaźnione

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class kwadrat{
5      private:
6          int a,b;
7      public:
8          SetAB(int x,int y)
9              {a=x;b=y;}
10 };
11 int main(){
12     kwadrat kw1;
13     kw1.SetAB(1,2);
14     return 0;
15 }
```

## Przypominamy:

- Do zmiennych private nie mamy dostępu z zewnątrz;
- Możemy robić to przez akcesory;
- Ale nie tylko... mamy funkcje zaprzyjaźnione.

# Funkcje zaprzyjaźnione

- Funkcje zaprzyjaźnione mają dostęp do wszystkich składowych klasy zadeklarowanych jako `private` lub `protected`;
- Z formalnego punktu widzenia łamią zasadę hermetyzacji (ukrywania) danych.

# Funkcje zaprzyjaźnione

## **Możliwe są następujące sytuacje:**

- Funkcja niezależna zaprzyjaźniona z klasą  $X$ ,
- Funkcja składowa klasy  $Y$  zaprzyjaźniona z klasą  $X$ ,
- Wszystkie funkcje klasy  $Y$  są zaprzyjaźnione z klasą  $X$  (klasa zaprzyjaźniona).

# Jak używamy funkcji zaprzyjaźnionych

```
1  #include <iostream>
2  #include <math.h>
3  #include <conio.h>
4  using namespace std;
5  class punkt{
6      float x, y ;
7      public:
8      void ustaw(float a ,float b);
9      friend float promien (punkt n);
10 };
11 void punkt::ustaw(float a, float b){
12     x = a ; y = b ;
13 }
14 float promien(punkt n){
15     return sqrt(n.x * n.x + n.y * n.y);
16 }
17 int main(){
18     punkt p1;
19     p1.ustaw(1.0,1.0) ;
20     cout << "Odleglosc = " << promien(p1) << endl ;
21     getch();
22     return 0;
23 }
```

Definicja klasy  
punkt



# Jak używamy funkcji zaprzyjaźnionych

```
1  #include <iostream>
2  #include <math.h>
3  #include <conio.h>
4  using namespace std;
5  class punkt{
6      float x, y ;
7      public:
8      void ustaw(float a ,float b);
9      friend float promien (punkt n);
10 };
11 void punkt::ustaw(float a, float b){
12     x = a ; y = b ;
13 }
14 float promien(punkt n){
15     return sqrt(n.x * n.x + n.y * n.y);
16 }
17 int main(){
18     punkt p1;
19     p1.ustaw(1.0,1.0) ;
20     cout << "Odleglosc = " << promien(p1) << endl ;
21     getch();
22     return 0;
23 }
```

**Deklaracja  
funkcji  
zaprzyjaźnionej  
(nie podlega  
specyfikatorom  
dostępu)**



# Jak używamy funkcji zaprzyjaźnionych

```
1  #include <iostream>
2  #include <math.h>
3  #include <conio.h>
4  using namespace std;
5  class punkt{
6      float x, y ;
7      public:
8      void ustaw(float a ,float b);
9      friend float promien (punkt n);
10 };
11 void punkt::ustaw(float a, float b){
12     x = a ; y = b ;
13 }
14 float promien(punkt n){
15     return sqrt(n.x * n.x + n.y * n.y);
16 }
17 int main(){
18     punkt p1;
19     p1.ustaw(1.0,1.0) ;
20     cout << "Odleglosc = " << promien(p1) << endl ;
21     getch();
22     return 0;
23 }
```

Definicja metody  
ustaw



# Jak używamy funkcji zaprzyjaźnionych

```
1  #include <iostream>
2  #include <math.h>
3  #include <conio.h>
4  using namespace std;
5  class punkt{
6      float x, y ;
7      public:
8      void ustaw(float a ,float b);
9      friend float promien (punkt n);
10 };
11 void punkt::ustaw(float a, float b){
12     x = a ; y = b ;
13 }
14 float promien(punkt n){
15     return sqrt(n.x * n.x + n.y * n.y);
16 }
17 int main(){
18     punkt p1;
19     p1.ustaw(1.0,1.0) ;
20     cout << "Odleglosc = " << promien(p1) << endl ;
21     getch();
22     return 0;
23 }
```

Definicja funkcji  
zaprzyjaźnionej  
promień - na  
zewnątrz



# Jak używamy funkcji zaprzyjaźnionych

```
1  #include <iostream>
2  #include <math.h>
3  #include <conio.h>
4  using namespace std;
5  class punkt{
6      float x, y ;
7      public:
8      void ustaw(float a ,float b);
9      friend float promien (punkt n);
10 };
11 void punkt::ustaw(float a, float b){
12     x = a ; y = b ;
13 }
14 float promien(punkt n){
15     return sqrt(n.x * n.x + n.y * n.y);
16 }
17 int main(){
18     punkt p1;
19     p1.ustaw(1.0,1.0) ;
20     cout << "Odleglosc = " << promien(p1) << endl ;
21     getche();
22     return 0;
23 }
```

wywołanie  
funkcji  
zaprzyjaźnionej  
promień

# Jak używamy funkcji zaprzyjaźnionych

```
1  #include <iostream>
2  #include <math.h>
3  #include <conio.h>
4  using namespace std;
5  class punkt{
6      float x, y ;
7      public:
8      void ustaw(float a, float b);
9      friend float promien (punkt n);
10 };
11 void punkt::ustaw(float a, float b){
12     x = a ; y = b ;
13 }
14 float promien(punkt n){
15     return sqrt(n.x * n.x + n.y * n.y);
16 }
17 int main(){
18     punkt p1;
19     p1.ustaw(1.0,1.0) ;
20     cout << "Odleglosc = " << promien(p1) << endl ;
21     getch();
22     return 0;
23 }
```

Odleglosc = 1.41421

# Klasy zaprzyjaźnione

- Każda klasa może mieć wiele funkcji zaprzyjaźnionych,
- Można także uczynić daną klasę zaprzyjaźnioną z inną klasą,
- Jeżeli klasa A jest uznawana za przyjaciela klasy B to oznacza, że wszystkie funkcje składowe klasy A mają dostęp do danych prywatnych i chronionych klasy B.

# Klasy zaprzyjaźnione

```
class B
{
    friend class A ;
    // .....
};
```

```
class A ;
```

# Klasy zaprzyjaźnione

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class Dane{
5      int x1,x2;
6      public:
7      Dane(int a,int b){x1=a;x2=b;}
8      friend class Test;
9  };
10 class Test{
11     public:
12     int min(Dane a){return a.x1<a.x2 ? a.x1:a.x2;}
13     int iloczyn(Dane a){return a.x1 * a.x2;}
14 };
15 int main(){
16     Dane liczby(10,20);
17     Test t;
18     cout << "mniejsza to: " << t.min(liczby) << endl;
19     cout << "    iloczyn    = " << t.iloczyn(liczby) << endl;
20     getch();
21     return 0;
22 }
```

# Klasy zaprzyjaźnione

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class Dane{
5      int x1,x2;
6      public:
7      Dane(int a,int b){x1=a;x2=b;}
8      friend class Test;
9  };
10 class Test{
11     public:
12     int min(Dane a){return a.x1<a.x2 ? a.x1:a.x2;}
13     int iloczyn(Dane a){return a.x1 * a.x2;}
14 };
15 int main(){
16     Dane liczby(10,20);
17     Test t;
18     cout << "mniejsza to: " << t.min(liczby) << endl;
19     cout << "    iloczyn    = " << t.iloczyn(liczby) << endl;
20     getch();
21     return 0;
22 }
```

Definicja klasy  
Dane

# Klasy zaprzyjaźnione

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class Dane{
5      int x1,x2;
6      public:
7      Dane(int a,int b){x1=a;x2=b;}
8      friend class Test; ←
9  };
10 class Test{
11     public:
12     int min(Dane a){return a.x1<a.x2 ? a.x1:a.x2;}
13     int iloczyn(Dane a){return a.x1 * a.x2;}
14 };
15 int main(){
16     Dane liczby(10,20);
17     Test t;
18     cout << "mniejsza to: " << t.min(liczby) << endl;
19     cout << "    iloczyn    = " << t.iloczyn(liczby) << endl;
20     getch();
21     return 0;
22 }
```

Określenie klasy  
zaprzyjaźnionej

# Klasy zaprzyjaźnione

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class Dane{
5      int x1,x2;
6      public:
7      Dane(int a,int b){x1=a;x2=b;}
8      friend class Test;
9  };
10 class Test{
11     public:
12     int min(Dane a){return a.x1<a.x2 ? a.x1:a.x2;}
13     int iloczyn(Dane a){return a.x1 * a.x2;}
14 };
15 int main(){
16     Dane liczby(10,20);
17     Test t;
18     cout << "mniejsza to: " << t.min(liczby) << endl;
19     cout << "    iloczyn    = " << t.iloczyn(liczby) << endl;
20     getch();
21     return 0;
22 }
```

Klasa  
zaprzyjaźniona  
Test



# Klasy zaprzyjaźnione

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class Dane{
5      int x1,x2;
6      public:
7      Dane(int a,int b){x1=a;x2=b;}
8      friend class Test;
9  };
10 class Test{
11     public:
12     int min(Dane a){return a.x1<a.x2 ? a.x1:a.x2;}
13     int iloczyn(Dane a){return a.x1 * a.x2;}
14 };
15 int main(){
16     Dane liczby(10,20);
17     Test t;
18     cout << "mniejsza to: " << t.min(liczby) << endl;
19     cout << "    iloczyn    = " << t.iloczyn(liczby) << endl;
20     getch();
21     return 0;
22 }
```

Operuje na  
danych  
prywatnych  
klasy Dane

# Przeciążanie operatorów

- Możemy tworzyć nowe typy ale nie możemy tworzyć nowych operatorów;
- Przeciążanie funkcji;
- Przeciążanie operatorów  
(W samym języku C++ mamy przeciążony np. operator dodawania – inaczej działa na różnych typach zmiennych)

# Przeciążanie operatorów

- Zmienia sposób działania operatora,
- Operacje wykonywane na obiektach typu klasa,
- Aby przeciążyć operator definiuje się funkcję, której nazwą jest słowo kluczowe operator i symbol operatora.

# Przeciążanie operatorów

- Realizowane w postaci osobnej funkcji,
- Realizowane w postaci funkcji składowej.

# Przykład dodawanie wektorów

$$a + b = c$$

$$ax + bx = cx$$

$$ay + by = cy$$

# Przykład

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class wek{
5      int vx , vy;
6      public :
7      void ustaw(int ux ,int uy);
8      void pokaz();
9      wek operator+ (wek v);
10 } ;
11 void wek::ustaw(int ux , int uy){
12     vx = ux ; vy = uy;
13 }
14 void wek::pokaz(){
15     cout<<"składowe wektora : ";
16     cout<<vx<<" , " <<vy<<endl;
17 }
```

```
20 wek wek::operator+(wek v){
21     wek wektor ;
22     wektor.vx=vx+v.vx;
23     wektor.vy=vy+v.vy;
24     return wektor;
25 }
26 int main()
27 {
28     wek a,b,suma_wek;
29     a.ustaw(1,1);
30     b.ustaw(1,2);
31     a.pokaz ();
32     b.pokaz ();
33     cout << "po dodaniu " ;
34     suma_wek = a + b ;
35     suma_wek.pokaz() ;
36     getch () ;
37     return 0 ;
}
```

# Przykład

## Klasa wektor

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class wek{
5      int vx , vy;
6      public :
7      void ustaw(int ux ,int uy);
8      void pokaz();
9      wek operator+ (wek v);
10 } ;
11 void wek::ustaw(int ux , int uy){
12     vx = ux ; vy = uy;
13 }
14 void wek::pokaz(){
15     cout<<"składowe wektora : ";
16     cout<<vx<<" , " <<vy<<endl;
17 }
```

```
20 wek wek::operator+(wek v){
21     wek wektor ;
22     wektor.vx=vx+v.vx;
23     wektor.vy=vy+v.vy;
24     return wektor;
25 }
26 int main()
27 {
28     wek a,b,suma_wek;
29     a.ustaw(1,1);
30     b.ustaw(1,2);
31     a.pokaz ();
32     b.pokaz ();
33     cout << "po dodaniu " ;
34     suma_wek = a + b ;
35     suma_wek.pokaz() ;
36     getch () ;
37     return 0 ;
}
```



# Przykład

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class wek{
5      int vx , vy;
6      public :
7      void ustaw(int ux ,int uy);
8      void pokaz();
9      wek operator+ (wek v);
10 } ;
11 void wek::ustaw(int ux , int uy){
12     vx = ux ; vy = uy;
13 }
14 void wek::pokaz(){
15     cout<<"składowe wektora : ";
16     cout<<vx<<" , " <<vy<<endl;
17 }
```

**Deklaracja funkcji  
przeciążonego operatora jako  
składowej klasy**

```
20
21
22
23
24
25 }
26 int main()
27 {
28     wek a,b,suma_wek;
29     a.ustaw(1,1);
30     b.ustaw(1,2);
31     a.pokaz ();
32     b.pokaz ();
33     cout << "po dodaniu " ;
34     suma_wek = a + b ;
35     suma_wek.pokaz() ;
36     getch () ;
37     return 0 ;
38 }
```

# Przykład

## Definicja funkcji przeciążonego operatora

```
1  #include
2  #include
3  using namespace std;
4  class wektor {
5      int vx, vy;
6      public:
7          void ustaw(int ux, int uy);
8          void pokaz();
9          wektor operator+ (wektor v);
10 } ;
11 void wektor::ustaw(int ux, int uy){
12     vx = ux; vy = uy;
13 }
14 void wektor::pokaz(){
15     cout<<"składowe wektora : ";
16     cout<<vx<<" , " <<vy<<endl;
17 }
```

```
20 wektor wektor::operator+(wektor v){
21     wektor wektor;
22     wektor.vx=vx+v.vx;
23     wektor.vy=vy+v.vy;
24     return wektor;
25 }
26 int main()
27 {
28     wektor a,b,suma_wektor;
29     a.ustaw(1,1);
30     b.ustaw(1,2);
31     a.pokaz ();
32     b.pokaz ();
33     cout << "po dodaniu ";
34     suma_wektor = a + b;
35     suma_wektor.pokaz();
36     getch ();
37     return 0;
}
```

# Przykład

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class wek{
5      int vx , vy;
6      public :
7      void ustaw(int ux ,int uy);
8      void pokaz();
9      wek operator+ (wek v);
10 } ;
11 void wek::
12     vx = u
13 }
14 void wek::
15     cout<<
16     cout<<vx<<" , " <<vy<<endl;
17 }
```

Wywołanie  
funkcji  
przeciążonego  
operatora

```
20 wek wek::operator+(wek v){
21     wek wektor ;
22     wektor.vx=vx+v.vx;
23     wektor.vy=vy+v.vy;
24     return wektor;
25 }
26 int main()
27 {
28     wek a,b,suma_wek;
29     a.ustaw(1,1);
30     b.ustaw(1,2);
31     a.pokaz ();
32     b.pokaz ();
33     cout << "po dodaniu " ;
34     suma_wek = a + b ;
35     suma_wek.pokaz() ;
36     getch () ;
37     return 0 ;
}
```

# Przykład

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class wek{
5      int vx , vy;
6      public :
7      void ustaw(int ux ,int uy);
8      void pokaz();
9      wek operator+ (wek v);
10 } ;
11 void wek::ustaw(int ux ,int uy){
12     Inny zapis:
13     suma_wek=a.operator+(b)
14     void wek::pokaz(){
15         cout<<"skladowe wektora : ";
16         cout<<vx<<" , " <<vy<<endl;
17     }
```

```
20 wek wek::operator+(wek v){
21     wek wektor ;
22     wektor.vx=vx+v.vx;
23     wektor.vy=vy+v.vy;
24     return wektor;
25 }
26 int main()
27 {
28     wek a,b,suma_wek;
29     a.ustaw(1,1);
30     b.ustaw(1,2);
31     a.pokaz ();
32     b.pokaz ();
33     cout << "po dodaniu " ;
34     suma_wek = a + b ;
35     suma_wek.pokaz();
36     getch ();
37     return 0 ;
}
```

# Przykład 2 - Operator mnożenia

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class ulamek{
5      int li,mia;
6      public:
7      set(int l,int m){li=l;mia=m;};
8      void pokaz(){cout<<li<<"/"<<mia<<endl;}
9      ulamek operator*(ulamek);
10 };
11 ulamek ulamek::operator*(ulamek x){
12     ulamek u;u.li=li * x.li; u.mia= mia * x.mia;
13     return u;
14 }
15 int main(){
16     ulamek u1,u2,u3;
17     u1.set(1,3);u2.set(2,7);u3.set(0,0);
18     u3=u1*u2;
19     u1.pokaz();u2.pokaz();u3.pokaz();
20     getch();
21 }
```

# Przykład 2 - Operator mnożenia

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class ulamek{
5      int li,mia;
6      public:
7      set(int l,int m){li=l;mia=m;};
8      void pokaz(){cout<<li<<"/"<<mia<<endl;}
9      ulamek operator*(ulamek);
10 };
11 ulamek ulamek::operator*(ulamek x){
12     ulamek u;u.li=li * x.li; u.mia= mia * x.mia;
13     return u;
14 }
15 int main(){
16     ulamek u1,u2,u3;
17     u1.set(1,3);u2.set(2,7);u3.set(0,0);
18     u3=u1*u2;
19     u1.pokaz();u2.pokaz();u3.pokaz();
20     getch();
21 }
```

1/3  
2/7  
2/21

# Przeciążanie

- operator przeciążony może być zdefiniowany jako funkcja składowa i jako funkcja globalna (lub zaprzyjaźniona),
- Jakie są kryteria wyboru implementacji?



# Przeciążanie

- **Jeżeli operator modyfikuje operandy to powinien być zdefiniowany jako funkcja składowa klasy. Przykładem są tu takie operatory jak: `=`, `+=`, `-=`, `*=`, `++`, itp.**
- **Jeżeli operator nie modyfikuje swoich operandów to należy go definiować jako funkcję globalną lub zaprzyjaźnioną.**

# Operator wstawiania <<

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class punkt{
5      int x , y ;
6      public :
7      punkt(int a,int b) {x = a;y = b;}
8      friend ostream& operator<< ( ostream &, punkt & );
9  };
10 ostream & operator<< (ostream & os , punkt & ob){
11     os << "x = " << ob.x << endl;
12     os << "y = " << ob.y << endl;
13     return os ;
14 }
15 int main(){
16     punkt p1(5,15);
17     cout << "wywołanie ukryte :  \n" ;
18     cout << p1;
19     cout << "wywołanie jawne :  \n" ;
20     operator<<(cout, p1) ;
21     getch();
22     return 0;
23 }
```

# Operator wstawiania <<

Zaprzyjaźniona  
funkcja  
przeciążonego  
operatora <<

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class punkt{
5      int x , y ;
6      public :
7      punkt(int a,int b) {x = a;y = b;}
8      friend ostream& operator<< ( ostream &, punkt & );
9  };
10 ostream & operator<< (ostream & os , punkt & ob){
11     os << "x = " << ob.x << endl;
12     os << "y = " << ob.y << endl;
13     return os ;
14 }
15 int main(){
16     punkt p1(5,15);
17     cout << "wywołanie ukryte : \n" ;
18     cout << p1;
19     cout << "wywołanie jawne : \n" ;
20     operator<<(cout, p1) ;
21     getch();
22     return 0;
23 }
```

# Operator wstawiania <<

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  class punkt{
5      int x , y ;
6      public :
7      punkt(int a,int b) {x = a;y = b;}
8      friend ostream& operator<< ( ostream &, punkt & );
9  };
10 ostream & operator<< (ostream & os , punkt & ob){
11     os << "x = " << ob.x << endl;
12     os << "y = " << ob.y << endl;
13     return os ;
14 }
15 int main(){
16     punkt p1(5,15);
17     cout << "wywołanie ukryte : \n" ;
18     cout << p1;
19     cout << "wywołanie jawne : \n" ;
20     operator<<(cout, p1) ;
21     getch();
22     return 0;
23 }
```

Wywołanie

# Operatory do przeciążania

Operator	Opis
()	Wywołanie funkcji (Function call)
[]	Element tablicy (Array element)
—>	Operator dostępu (Structure member pointer reference)
New	Dynamicznie alokowana pamięć (Dynamically allocate memory)
Delete	Dynamicznie usuwana pamięć (Dynamically deallocated memory)
++	Inkrementacja (Increment)
—	Dekrementacja (Decrement)
-	Minus (Unary minus)
!	Logiczna negacja (Logical negation)
~	Dopełnienie logiczne (One's complement)
*	Dereferencja (Indirection)
*	Mnożenie (Multiplication)
/	Dzielenie (Division)
%	Modulo (Modulus (remainder))

# Operatory do przeciążania

+	Dodawanie (Addition)
-	Odejmowanie (Subtraction)
<<	Przesunięcie (Left shift)
>>	Przesunięcie (Right shift)
<	Mniej niż (Less than)
<=	Mniej niż lub równe (Less than or equal to)
>	Większe niż (Greater than)
>=	Większe niż lub równe (Greater than or equal to)
==	Równe (Equal to)
!=	Różne (Not equal to)
&&	Logiczne AND (Logical AND)
	Logiczne OR (Logical OR)
&	Bitowe AND (Bitwise AND)
^	Bitowe XOR (Bitwise exclusive OR)
	Bitowe OR (Bitwise inclusive OR)



# Operatory do przeciążania

=	Przypisanie (assignment)
+= -= *=	Przypisanie (assignment)
/= %= &=	Przypisanie (assignment)
^=  =	Przypisanie (assignment)
<<= >>=	Przypisanie (assignment)
,	Przecinek (Comma)



# Typ wyliczeniowy

- Tworzony za pomocą słowa kluczowego enum,
- Jest to skończony zbiór pewnych elementów,
- Zadaniem typu wyliczeniowego jest zwiększenie czytelności programu.

# Typ wyliczeniowy

- **enum** *nazwa\_typu* { *lista\_wyliczenia* }  
*lista\_zmiennych*
- Elementy wyliczeniowe umieszczane są w nawiasie klamrowym,
- Domyślną wartością pierwszego elementu jest „0”, a każdy następny element w liście otrzymuje wartość o jeden większą od poprzedniego,
- *nazwa\_typu*, *lista\_zmiennych* są *opcjonalne*,

# Typ wyliczeniowy

enum { fałsz, prawda }



enum { fałsz, nic = 0, sukces, prawda = 1 }



Czym różni się typ wyliczeniowy od stałych?

# Typ wyliczeniowy

- Dla elementu wyliczeniowego nie przydziela się adresowalnego obszaru pamięci.

# Przykład

Jaki będzie wynik ?

```
#include <stdio.h>
#include <conio.h>
#include <iostream>
using namespace std;

enum kolor {
    bialy, czarny, zielony, inny=10, inny_niz_inny
};

int main(){
    cout << "bialy = " << bialy << endl;
    cout << "czarny =" << czarny << endl;
    cout << "zielony =" << zielony << endl;
    cout << "inny =" << inny << endl;
    cout << "inny_niz_inny =" << inny_niz_inny << endl;
    getch();
    return 0;
}
```

# Przykład

```
biały = 0  
czarny = 1  
zielony = 2  
inny = 10  
inny_niz_inny = 11
```

# Przykład 2

Jak to zadziała ?

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  string dni_tab[]={"poniedzialek","wtorek","sroda",
5  "czwartek","piatek","sobota","niedziela"};
6  enum dni {pon,wto,sro,czw,pia,sob,nie};
7  int main(){
8      dni dzisiaj=pon;
9      dni jutro=wto;
10     cout << "Dzisiaj jest " << dni_tab[dni(dzisiaj)] << endl;
11     cout << "Jutro bedzie " << dni_tab[dni(jutro)] << endl;
12     cout << "pon = " << pon <<endl;
13     cout << "wto = " << wto <<endl;
14     cout << "sro = " << sro <<endl;
15     cout << "czw = " << czw <<endl;
16     cout << "pia = " << pia <<endl;
17     cout << "sob = " << sob <<endl;
18     cout << "nie = " << nie <<endl;
19 }
```



## Przykład 2

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  string dni_tab[]={"poniedzialek","wtorek","sroda",
5  "czwartek","piatek","sobota","niedziela"};
6  enum dni {pon,wto,sro,czw,pia,sob,nie};
7  int main(){
8      dni dzisiaj=pon;
9      dni jutro=wto;
10     cout << "Dzisiaj jest poniedzialek";
11     cout << "Jutro bedzie wtorek";
12     cout << "pon = " << pon;
13     cout << "wto = " << wto;
14     cout << "sro = " << sro;
15     cout << "czw = " << czw;
16     cout << "pia = " << pia;
17     cout << "sob = " << sob;
18     cout << "nie = " << nie;
19 }
```

Dzisiaj jest poniedzialek  
Jutro bedzie wtorek  
pon = 0  
wto = 1  
sro = 2  
czw = 3  
pia = 4  
sob = 5  
nie = 6

# Przykład 3

Jak to zadziała ?

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  enum dni {pon=9,wto,sro,czw,pia=5,sob,nie};
5  int main(){
6      cout << "pon = " << pon <<endl;
7      cout << "wto = " << wto <<endl;
8      cout << "sro = " << sro <<endl;
9      cout << "czw = " << czw <<endl;
10     cout << "pia = " << pia <<endl;
11     cout << "sob = " << sob <<endl;
12     cout << "nie = " << nie <<endl;
13 }
```

## Przykład 3

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  enum dni {pon=9,wto,sro,czw,pia=5,sob,nie};
5  int main(){
6      cout << "pon = " << pon;
7      cout << "wto = " << wto;
8      cout << "sro = " << sro;
9      cout << "czw = " << czw;
10     cout << "pia = " << pia;
11     cout << "sob = " << sob;
12     cout << "nie = " << nie;
13 }
```

```
pon = 9
wto = 10
sro = 11
czw = 12
pia = 5
sob = 6
nie = 7
```



# KONIEC