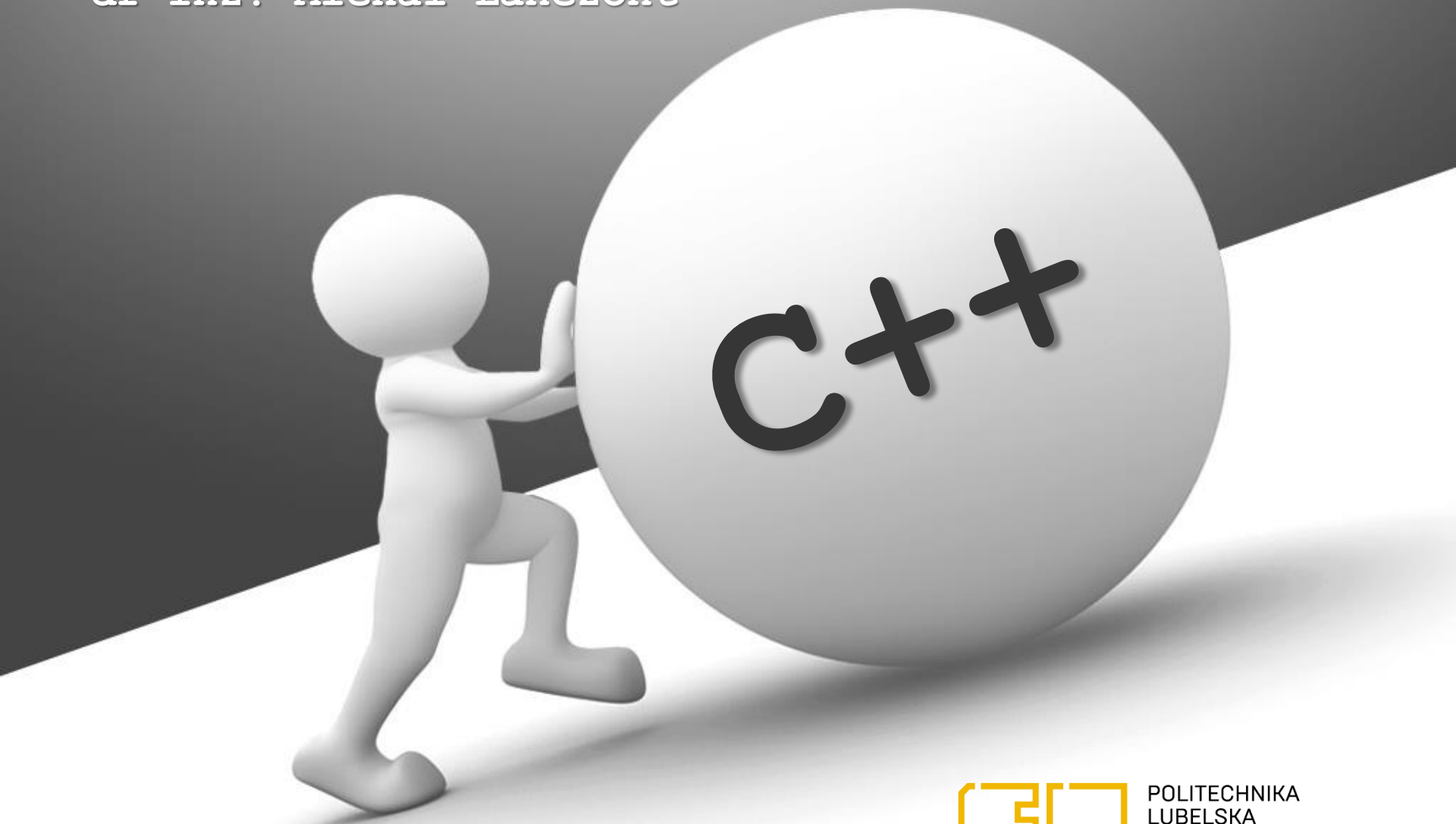
Informatyka II

dr inż. Michał Łanczont



POLITECHNIKA
LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI
I INFORMATYKI

Zakres wykładu

1. Przeciążanie
   - *A. Funkcji*
   - *B. Operatorów*
2. Funkcje i klasy zaprzyjaźnione
3. Dziedziczenie
4. Polimorfizm
   - *A. Funkcje wirtualne*
5. Szablony
   - *A. Funkcji*
   - *B. Klas*
6. Wyrażenia regularne

Przeciążanie funkcji

- Deklarując funkcje lub metodę określa się jej unikalną nazwę, typ zwracanej wartości oraz typy parametrów wejściowych

- Może się jednak zdarzyć, że to samo działanie będziemy chcieli wykonywać dla kilku różnych typów zmiennych

Przeciążanie funkcji

- W efekcie trzeba by tworzyć kilka funkcji realizujących niemalże to samo zadanie ale dla innych typów zmiennych
- C++ umożliwia zdefiniowanie kilku funkcji różniących się tylko typami zadeklarowanych zmiennych wejściowych i wyjściowych, ale posiadających tą samą nazwę funkcji
- Procedurę tą określa się **przeciążaniem funkcji**

Przeciążanie funkcji

- W programie lub klasie deklaruje się funkcje lub metody o tej samej nazwie
- Funkcje te mogą:
  - Zwracać wartości różnych typów
  - Parametry wejściowe mogą być różnych typów
  - Mogą się różnić liczbą parametrów wejściowych
  - Mogą wykonywać różne zadania – różnić się nawet znacząco kodem

Przykład

```
15  int random(int P,int K)
16  {
17      return P + rand()%(K-P+1);
18  }
19  float random(int n){
20      if(n>5)
21      {
22          cout << "Maksymalna dokładność n=4";
23          return 0;
24      }
25      int m = pow(10,n);
26      float R = rand()%(m+1);
27      return R/m;
28  }
29  char random()
30  {
31      return 65 + rand()%(90-65+1);
32  }
```

```
C:\Windows\system32\cmd.exe
random(20,30) = 27
random(3) = 0.905
random() = Y

Press any key to continue . . .
```

Przeciążanie operatora

- Przeciążenie, czyli nadanie kolejnej funkcjonalności można nadać nie tylko funkcjom lub metodom w klasie, ale także operatorom.
- Operatory w języku C++ spełniają różne, często odmienne role w różnych sytuacjach
- < jest znakiem „mniejsze niż” ale także << jako znak przesłania danych do standardowego strumienia wyjściowego

Przeciążanie operatora

- Deklarację przeciążenia operatora (+,-,*,/,%,<,>,…) umieszcza się w deklaracji klasy

+	-	*	/	%	^	&
|	_	!	=	<	>	+=
-=	*=	/=	%=	^=	&=	|=
<<	>>	||	++	--	->*	,
->	[]	()	new	new[]	delete	delete[]

Przeciążanie operatora

- Operator dodawania + jest w języku C++ operatorem przeciążonym przez wiele klas
- Poszczególne typy zmiennych (int, float, double, string, complex)
- W każdej z nich operator ten realizuje inne zadania

Przeciążanie operatora

- Przeciążenie operatora deklaruje się w klasie za pomocą słowa kluczowego operator
- Deklaruje się ją podobnie jak funkcje
- Typem wyjściowym jest klasa w której jest deklarowane
- Atrybutem wejściowym deklaracji jest obiekt tej samej klasy

07:46

Przykład

```
#include <iostream>
#include <ctime>
using namespace std;
class liczby{
private:
    int *num;
    int n;
public:
    liczby(int
    liczby()…
    ~liczby()…
    void losuj(
    void print()…
    void operator << (int n)…
};
int main()
{
    liczby A,B(5);
    cout << "A = "; A.print(); cout << endl;
    cout << "B = "; B.print(); cout << endl;
    A << 10;
    cout << "A << 10 = "; A.print(); cout << endl;
    B << 1;
    cout << "B << 1 = "; B.print(); cout << endl;
    A << 0; B << -1;
}
```

```
    {
        n=1;
        num = new int;
        srand(time(0));
        num[0]=0;
    }
```

C:\WINDOWS\system32\cmd.exe

```
A = 0
B = 0 0 0 0 0
A << 10 = 91 37 35 6 12 64 9 27 86 90
B << 1 = 37
Nie można ustawić rozmiaru tablicy na: 0
Nie można ustawić rozmiaru tablicy na: -1

Press any key to continue . . .
```

Przyjaźń

- Do pól i metod prywatnych dostęp ograniczony jest dla metod klasy
- Czasem konieczne jest, aby z poziomu programu uzyskać dostęp dostęp do pól i metod prywatnych klasy
- Sytuacja ta ma miejsce gdy operujemy na wielu obiektach danej klasy lub przetwarzamy obiekty powstałe na bazie różnych klas

Funkcje zaprzyjaźnione

- W języku C++ jest mechanizm dopuszczający wskazaną funkcję do elementów prywatnych klasy
- W klasie można zdefiniować „przyjaźń” pomiędzy klasą i funkcją
- Funkcja może być „zaprzyjaźniona” z wieloma klasami, a klasa z wieloma funkcjami
- Parametrem wejściowym funkcji są obiekty klas „zaprzyjaźnionych”

Funkcje zaprzyjaźnione

- To klasa określa jaka funkcja będzie z nią „zaprzyjaźniona”
- Przyjaźń deklaruje się w obszarze `public:`
- Funkcję „zaprzyjaźnioną” należy zadeklarować poprzedzając słowem kluczowym `friend`

Funkcja zaprzyjaźniona

```
class nazwa
{
private:
   typ x,y,z;
public:
   nazwa();
   void func(typ);
   friend void funcZew(nazwa);
};
```

Przykład

```
39  bool test(punkt P,kolo K)
40  {
41      float dist;
42      dist = sqrt(pow(P.x-K.x,2)+pow(P.y-K.y,2));
43      cout << fixed << setprecision(2);
44      cout << ":dystans:" << dist << ":";
45      if(dist > K.r)
46          return false;
47      return true;
48  }
27          }
```

C:\Windows\system32\cmd.exe

```
Czy P1 jest w K: :dystans:14.14:false
Czy P2 jest w K: :dystans:7.07:true

Press any key to continue . . .
```

```
40 › bool test(punkt P,kolo K)···
```

Klasa zaprzyjaźniona

- Podobnie jak w przypadku funkcji możliwe jest ustawienie klasy jako zaprzyjaźnionej
- Przyjaźń należy zadeklarować w klasie która ma nadać dostęp do cech i metod prywatnych
- W efekcie obiekty utworzone na bazie klasy zaprzyjaźnionej uzyskują dostęp do atrybutów i metod prywatnych klasy deklarującej przyjaźń

Przykład

```
44  bool test(punkt P,kolo K)
45  {
46      float dist;
47      dist = sqrt(pow(P.x-K.P->x,2)+pow(P.y-K.P->y,2));
48      cout << fixed << setprecision(2);
49      cout << ":dystans:" << dist << ":";
50      if(dist > K.r)
51          return false;
52      return true;
53  }
28          }
```

```
C:\Windows\system32\cmd.exe
Czy P1 jest w K: :dystans:14.14:false
Czy P2 jest w K: :dystans:7.07:true

Press any key to continue . . .
```

```
44 › bool test(punkt P,kolo K)…
```

Dziedziczenie

- Mechanizm współdzielenia funkcjonalności pomiędzy klasami
- Rozwinięcie funkcjonalności w stosunku do przyjaźni
- Przyjaźń wymaga utworzenia obiektu
- Klasa dziedzicząca poza swoimi atrybutami i metodami posiada także atrybuty i metody klasy po której dziedziczy
- Zapewnia mechanizm ponownego użycia kodu

Dziedziczenie

- Potomek (klasa pochodna) rozszerza możliwości rodzica (klas podstawowa)
- Klasa może dziedziczyć po więcej jak jednej klasie
- W efekcie nie ma konieczności ponownego pisania kodu wykonującego tą samą czynność

Dziedziczenie

- Dzięki dziedziczeniu raz utworzony kod może być wielokrotnie wykorzystywane w wielu programach
- Utworzone klasy mogą stanowić bazę, bibliotekę gotowych rozwiązań
- Dostosowanie istniejącej klasy do nowego zastosowania dzięki dziedziczeniu staje się stosunkowo prostym zadaniem
- Tworzy się klasę dziedziczącą wymagane funkcjonalności z istniejących klas i dokłada się nowe dostosowujące nową klasę do realizowanego zadania.

Dziedziczenie

- Klasa pochodna może także zostać klasą podstawową, z której będzie dziedziczyła inna klasa
- Jest to tzw. dziedziczenie wielopokoleniowe
- Tworzenie nowej klasy na bazie jednej klasy – dziedziczenie jednokrotne
- Tworzenie nowej klasy na bazie kilku klas – dziedziczenie wielokrotne

Dziedziczenie

- Jeżeli w klasie pochodnej i podstawowej istnieją atrybuty o tych samych nazwach to atrybut w klasie pochodnej „przesłoni” atrybut z klasy podstawowej
- Wykorzystując operator zasięgu „::” można w takiej sytuacji odwołać się do atrybutu klasy podstawowej
- Analogiczna sytuacja zachodzi także dla metod

Dziedziczenie

- Poziomy dostępu do atrybutów i metod z klasy podstawowej
- `public:` klasa ma dostęp do wszystkich metod i atrybutów
- `private:` zostają odziedziczone, ale w zakresie klasy pochodnej nie ma do nich bezpośredniego dostępu
- `protected:` metody i atrybuty są dostępne tylko dla klasy pochodnej

Dziedziczenie

- Deklarując dziedziczenie także określa się jego poziom dla funkcji które będą korzystały z klasy pochodnej
- Wybierając poziom decydujemy o sposobie dostępu tych funkcji do dziedziczonych elementów

Dziedziczenie

- Dziedziczenie public: dziedziczone elementy public i protected pozostają na tym samym poziomie
- Dziedziczenie protected: dziedziczone elementy public i protected są widziane jako protected
- Dziedziczenie private: dziedziczone elementy public i protected są widziane jako private

Dziedziczenie

		Sposób dziedziczenia		
		Public:	Protected:	Private:
Widoczność w klasie bazowej	Public:	Public:	Protected:	Private:
	Protected:	Protected:	Protected:	Private:
	Private:	Niedostępne	Niedostępne	Niedostępne

Niedostępne: o ile nie ustawiono przyjaźni.

Dziedziczenie

- W deklaracji klasy pochodnej należy dodać informacji klasie podstawowej i poziomie dostępu do niej

```
class prostokat: public punkt
  {

  };
```

Dziedziczenie

- **Po klasie podstawowej klasa pochodna nie dziedziczy:**
  - konstruktorów,
  - destruktorów,
  - operatora przypisania.

Przykład

```
18 class dziedzicz : protected baza
19 {
20 public:
21     void print()
22     {
23         co
24         co
25         //
26     }
27 };
28 int main()
29 {
30     dziedzicz A;
31     A.print();
32     //cout << A.pub << endl;
33 }
```

```
ublic baza
                                    ;
                                    l;
                                    ndl;
```

kod7

```
PUBLIC
PROTECTED

Press any key to continue...
```

```
32     cout << A.pub << endl;
33 }
```

Przykład

```
39  class kwadrat : public kolo
40  {
41  private:
42      int a;
43
44
45
46
47
48
49
50
51
52
53
54
55      {
56          cout << nazwa << "[" << x << "," << y << "](" << r << ")";
57          cout << "\n\t:dł. boku kwadratu wpisanego -> " << b;
58          cout << "\n\t:dł. boku kwadratu opisanego -> " << a << endl;
59      }
60  };
```

kod5

```
Alfa[13,14]
Beta[15,52]
Beta[15,52](25)
Delta[10,10](30)
        :dł. boku kwadratu wpisanego -> 42.4264
        :dł. boku kwadratu opisanego -> 60

Press any key to continue...
```

```
60  };
```

Polimorfizm

- Inaczej WIELOPOSTACIOWOŚĆ
- Możliwość przyjmowania przez funkcje i metody (w klasach) wielu form
- Polimorfizm STATYCZNY – przeciążanie funkcji i metod

- Polimorfizm DYNAMICZNY – powiązane z dziedziczeniem wyabstrahowanie metod od konkretnego typu danych
  - Metody wirtualne
  - Klasy abstrakcyjne

Polimorfizm

- Polimorfizm statyczny i dynamiczny poprawia stopień uniwersalności i czytelności kodu oraz ułatwia skalowanie kodu

- Pod pojęciem skalowalności można rozumieć łatwość rozbudowywania kodu o nowe funkcjonalności bez konieczności znaczącego ingerowania w już istniejący

Polimorfizm

- Polimorfizm dynamiczny opiera się na bazowej klasie abstrakcyjnej
- Wykorzystuje się efekt przesłaniania metod z klasy bazowej przez metody z klasy dziedziczącej
- Przy polimorfizmie ma miejsce efekt odwrotny, poprzez przesłoniętą metodę odwołuje się do metody przykrywającej
- Klasa abstrakcyjna zawiera wirtualne metody wskazujące na metody w klasach dziedziczących

Polimorfizm

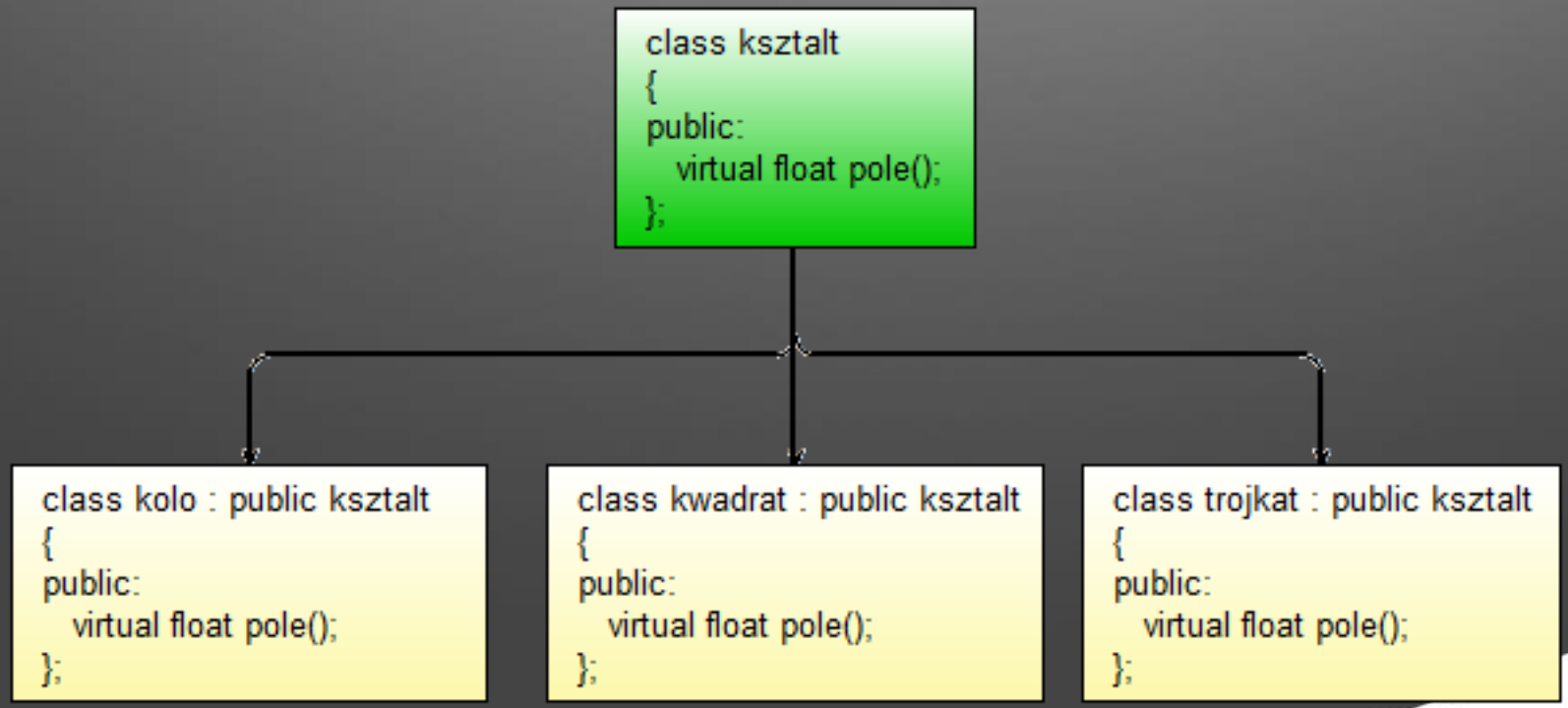
- Metody wirtualne są puste – nie zawierają żadnego kodu
- W klasa dziedziczących są metody o tej samej składni i realizujące kod przewidziany dla nich w danej klasie
- Deklarację tych metod poprzedza się słowem virual
- W klasie abstrakcyjnej (bazowej)

```
virual typ nazwa(typ,…)=0;
```

Polimorfizm

Odwołanie do metod wirtualnych realizowane jest przez wskaźnik polimorficzny

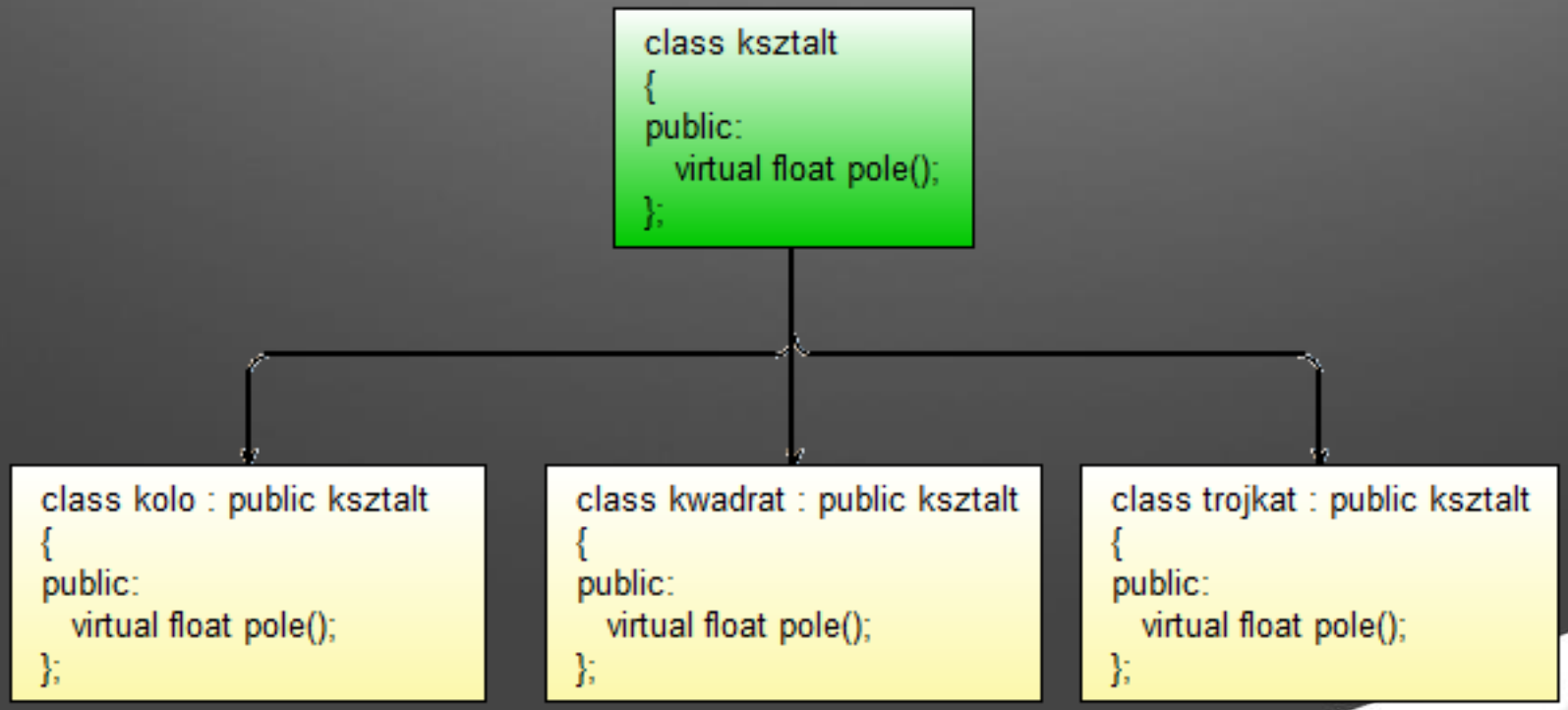
```
kształt *wskVirt;
```



Polimorfizm

Dynamiczne wywołanie metody poprzez wskaźnik polimorficzny

```
kolo A;
wskVirt = &A;
wskVirt->pole();
```



Przykład 1

```
1  #include <iostream>
2  #include <iomanip>
3  #include <cmath>
4  using namespace std;
5  class ksztalt…
10 class kolo : public ksztalt…
23 class kwadrat : public ksztalt…
5  class ksztalt
6  {
7      public:
8          virtual float pole()=0;
9  };
10 class kolo : public ksztalt
11 {
12     float r;
13     public:
14         kolo(float r)
15         {
16             this->r=r;
17         }                          lt *figura)
18         virtual float pole()
19         {                            ";
20             return M_PI*r*r;
21         }                          gura->pole() << endl;
22 };
```

```
polimorf
Pole    KOŁO : 78.54
Pole KWADRĄT : 25
Pole TRÓJKĄT : 12.5

Press any key to continue...
```

07:46

Przykład 2

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
class ksztalt…
class kolo : public ksztalt…
class kwadrat : public ksztalt…
class trojkat : public ksztalt…
void showPole(string, ksztalt*);
int main()
{
    ksztalt *A = new kolo(5);
    ksztalt *B = new kwadrat(5);
    ksztalt *C = new trojkat(5,5);

    ksztalt *ks[3] = {A,B,C};
    string nazwa[3] = {"KOLO","KWADRAT","TROJKAT"};
    for(int i=0;i<3;i++)
        showPole(nazwa[i],ks[i]);
}
void showPole(string nazwa, ksztalt *figura)
{
    cout << "Pole " + nazwa + " : ";
    cout.fixed;
    cout << setprecision(4) << figura->pole() << endl;
}
```

```
C:\Windows\system32\cmd.exe
Pole KOLO : 78.54
Pole KWADRAT : 25
Pole TROJKAT : 12.5

Press any key to continue . . .
```

Polimorfizm

- Polimorfizm można stosować także wtedy gdy metoda virtualna z klasy bazowej nie jest metodą pustą a klasa abstrakcyjną

- Metoda virtualna z klasy bazowej do której zawsze się odwołujemy wykonywana jest wtedy gdy w klasie dziedziczącej nie ma odpowiedniej metody

Przykład 1

```
38  class mlotek : public towar
39  {
40      float waga, dl;
41      str
42  public:
43      mlo                                          ring material)
44
45
46
47
48
49
50
51
52  };
33      {
34
35                                                   " cm" << endl;
36      }
37  };
```

C:\Windows\system32\cmd.exe

```
Anna: 1.34 zł.
Anna: plaski, 3 cm
------------------------------
Monika: 1.38 zł.
Monika: kulka, 4 cm
------------------------------
Mietek: 22.54 zł
------------------------------
Albert: 50 zł
------------------------------

Press any key to continue . . .
```

```
64  }
```

07:46

Przykład 2

```
19  class g
20  {
21      int
22      str
23  public:
24      gwc
25      {
26
27
28      }
29      vir
30      {
31
32
33      }
34  };
```

```
31      }
32  };
```

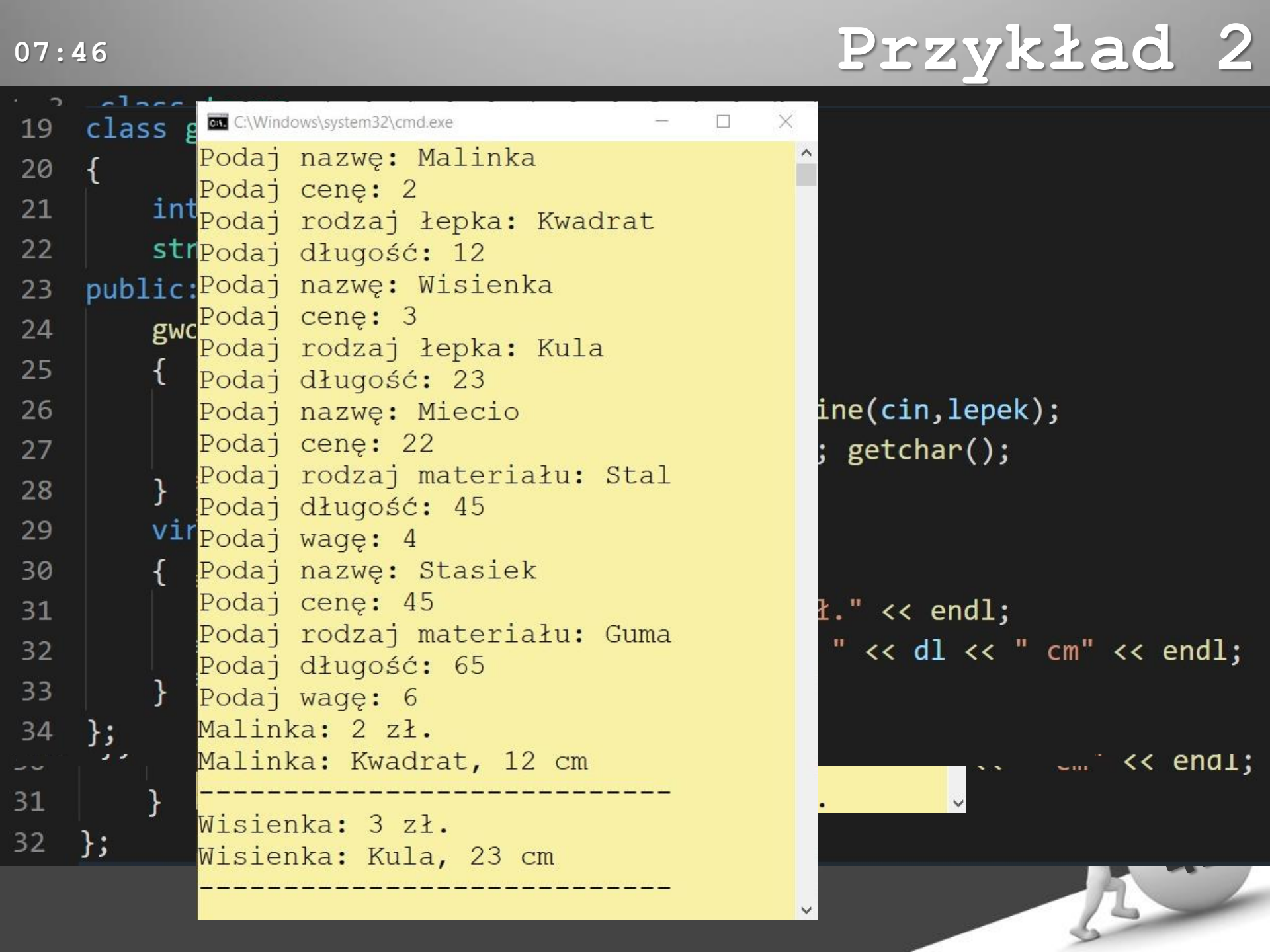
```
ine(cin,lepek);
; getchar();

ł." << endl;
" << dl << " cm" << endl;

<< endl;
```

C:\Windows\system32\cmd.exe

```
Podaj nazwę: Malinka
Podaj cenę: 2
Podaj rodzaj łepka: Kwadrat
Podaj długość: 12
Podaj nazwę: Wisienka
Podaj cenę: 3
Podaj rodzaj łepka: Kula
Podaj długość: 23
Podaj nazwę: Miecio
Podaj cenę: 22
Podaj rodzaj materiału: Stal
Podaj długość: 45
Podaj wagę: 4
Podaj nazwę: Stasiek
Podaj cenę: 45
Podaj rodzaj materiału: Guma
Podaj długość: 65
Podaj wagę: 6
Malinka: 2 zł.
Malinka: Kwadrat, 12 cm
----------------------------
Wisienka: 3 zł.
Wisienka: Kula, 23 cm
----------------------------
```



Szablon funkcji

- Przeciążanie funkcji pozwala na zapisanie w kodzie wielu funkcji realizujących to samo zadanie dla różnych typów danych
- Wymaga to zaprogramowania wielu funkcji
- Rozwiązaniem było by zaprogramowanie funkcji z adaptacyjnym typem danych
- Można to osiągnąć pisząc kod funkcji z szablonem typu zmiennej

Szablon funkcji

- **Funkcję generyczną (szablon funkcji) należy poprzedzić deklaracją szablonu typów stosowanych w kodzie**

```
template<class nazwa>
template<typename nazwa>
```

- **Szablon można zdefiniować dla kilku typów danych**

```
template<class T1, class T2>
template<typename T1, typename T2>
```

Szablon funkcji

- Zadeklarowany szablon typów może być stosowany w deklaracji i kodzie powiązanej z nim funkcji
- Typ **zwracany** przez funkcję generyczną musi być przypisany także do **co najmniej** jednego parametru wejściowego

```
template<class nowyTyp>
nowyTyp modul(nowyTyp A)
{
   return A<0 ? -A : A;
}
```

Szablon funkcji

Wywołanie funkcji generycznej możliwe jest na dwa sposoby:

- Wskazanie typów przy wywołaniu funkcji

```
x = modul<float>(liczba);
```

- Automatyczna detekcja na podstawie typów danych przekazywanych do funkcji

```
x = modul(liczba);
```

Przykład

```
kod9
1 : 10.11
2 : 12
3 : ?
4 : 9.75
5 : 11
6 : 11.86

Press any key to continue...
```

```
name typ2>
```

```
 9  {
10      float a=5.75, b=6.11;
11      int A=4, B=8;
12      cout << 1 << " : " << suma<float,float>(A,b) << endl;
13      cout << 2 << " : " << suma<int,int>(A,B) << endl;
14      cout << 3 << " : " << suma('A','a') << endl;
15      cout << 4 << " : " << suma(a,A) << endl;
16      cout << 5 << " : " << suma<int,int>(a,b) << endl;
17      cout << 6 << " : " << suma(a,b) << endl;
18  }
32  }
```

Szablon klasy

- Szablon może zostać także „podpięty” do klasy

- Obiekt utworzony na bazie takie klasy będzie obiektem uniwersalnym, adaptującym się do wybranych typów

- Wybór typu/ów przypisanych do szablonu klasy realizowany jest w momencie tworzenia obiektu

Szablon klasy

- **Deklaracja szablonu klasy poprzedza kod klasy**

```
template<typename nowyTyp>
```

lub

```
template<class nowyTyp>
```

- **W tworzonej klasie cechy i metody można oprzeć o typ szablonu**

```
class nazwaKlasy{};
```

Szablon klasy

- Tworzenie obiektu na bazie szablonu klasy wymaga określenia typów szablonu
- Klasa z szablonem nie potrafi przypisać typów z konstruktora do typów szablonowych

```
klasa<typDanych>(zmienna,…);
```

Przykład

```
1  #include <iostream>
2  using namespace std;
3  template<class zm1, class zm2>
4  class punkt
5  {
```

C:\Windows\system32\cmd.exe

```
Alfa.[5,5];
Beta.[A,B];
Press any key to continue . . .
```

```
21 int main()
22 {
23     punkt<string,int> pierwszy("Alfa",5,5);
24     punkt<string,char> drugi("Beta",'A','B');
25     pierwszy.print();
26     drugi.print();
27 }
```

Compiled successfully!

Wyrażenie regularne

- Przetwarzane w programach dane (tekstowe) powiązane z określonymi typami informacji podlegają określonym wymogom składniowym
  - Numer telefonu
  - Pesel
  - Adres email
  - NIP
  - Kod pocztowy
  - Hasło
- Analiza i weryfikacja danych tego typu w klasycznym podejściu jest zadaniem bardzo złożonym

Wyrażenie regularne

- W języku C++ dostępna jest biblioteka zawierająca narzędzia dedykowane do weryfikacji składniowej wprowadzanego ciągu znakowego
- **Regular expression – REGEX**
- Dostarcza narzędzi dokładnego identyfikowania określonego ciągu tekstowego i wykonywania określonych operacji
- Narzędzia `regex` są w przestrzeni nazw `std`

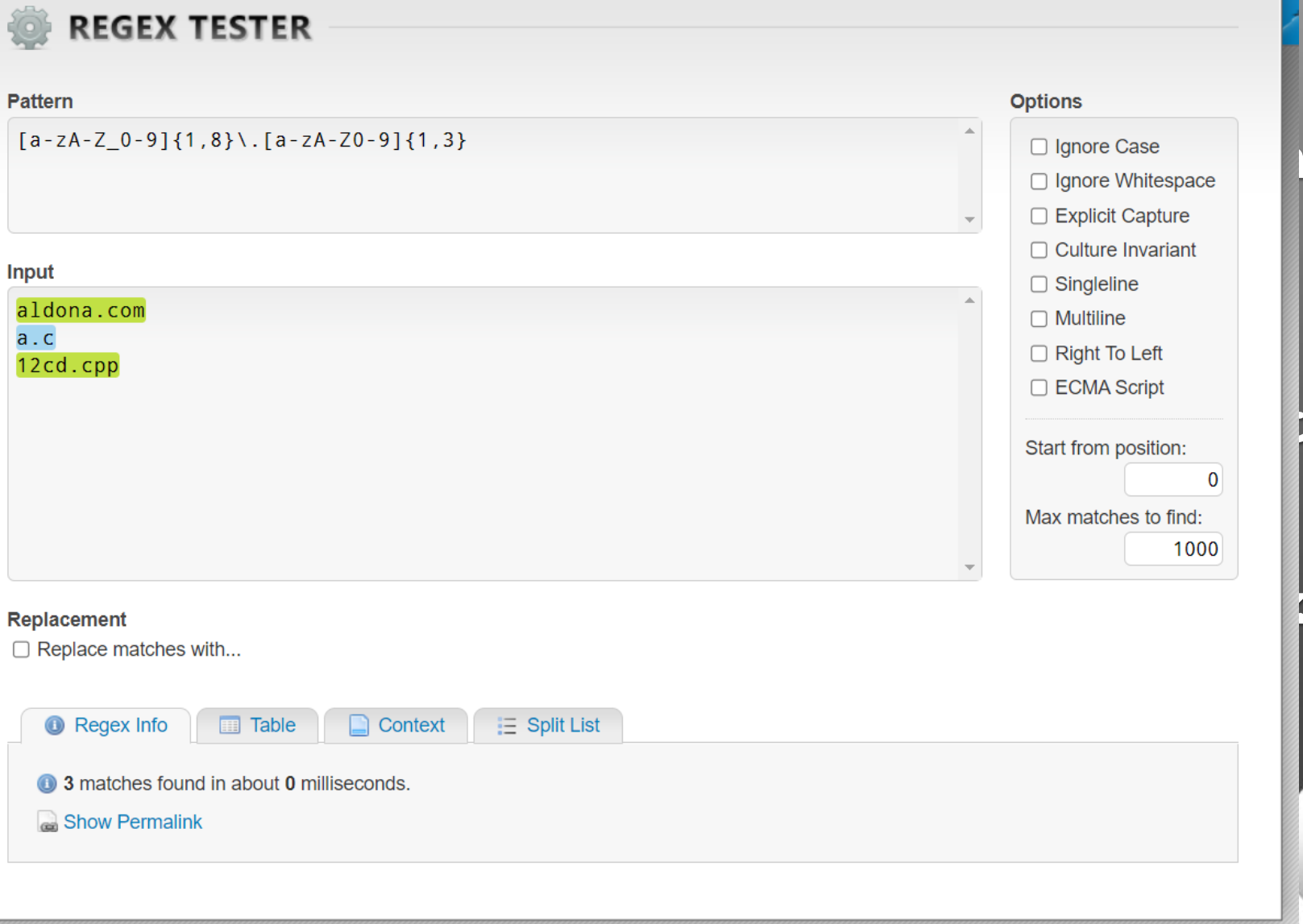
Wyrażenie regularne

1. Porównanie ciągu tekstowego z zakodowanym wzorcem

2. Wyszukiwanie określonego za pomocą wzorca ciągu znakowego w tekście

3. Zastąpienie w ciągu tekstowym znaków lub ciągów tekstowych opisanych wzorcem w tekście na podany ciąg znakowy

Wyrażenie regularne



Wyrażenie regularne

- Szablon wyrażenia regularnego definiuje się za pomocą odpowiednich znaków formatujących w postaci ciągu tekstowego przypisanego do dedykowanej w bibliotece zmiennej obiektowej `regex`
- Szablon może być przypisany w momencie tworzenia obiektu lub przypisany w dowolnym momencie

Wyrażenie regularne

Znaki formatujące stosowane do budowy wzorca

.	Dowolny jeden znak
[[:klasa:]]	Nazwa klasy znaków
{**n**}	Liczba znaków
(wyrażenie)	Złożone wyrażenie
****znak	Po ‚\' znak specjalny (. ? + * $)
wyrażenie*****	Zero lub więcej znaków
wyrażenie**+**	Jeden lub więcej znaków
wyrażenie**?**	Zero lub jeden znak
wyr1**|**wyr2	Lub (OR)
^wyr	Negacja (!)
$	Wzorzec jest na końcu wiersza

Wyrażenie regularne

Determinacja znaków dla wyrażenia

Znak	Negacja	Znaczenie
\d	\D	Cyfra dziesiętna
\l	\L	Mała litera
\s	\S	Biały znak (spacja, tabulacja)
\u	\U	Wielka litera
\w	\W	Litera, cyfra lub „ „

Wyrażenie regularne

Liczba znaków w wyrażeniu

{n}	Dokładnie **n** razy
{n,}	Co najmniej **n** znaków
{n,m}	Co najmniej **n** i nie więcej jak **m** znaków
*	{0,}
+	{1,}
?	{0,1}

Wyrażenie regularne

Opis wyrażenia można zapisać w postaci zbiorów i przedziałów ujętych w []

[r-\w]	Mała litera od **r** do **w**
[a &]	Litera **a**, **spacja** lub znak **&**
[a-zA-Z]	Litera alfabetu mała lub duża od **a** do **z**
[^abc]	Dowolny znak poza **a**, **b** i **c**

Wyrażenie regularne

Do opisu wyrażenia można wykorzystać zdefiniowane w bibliotece klasy

alnum	dowolny znak alfanumeryczny (\w)
alpha	dowolny znak alfabetu (bez polskich znaków)
blank	dowolny biały znak oprócz \n
cntrl	dowolny znak sterujący (np. [Ctrl] + [a])
d	dowolna cyfra dziesiętna(\d)
digit	jak wyżej
graph	dowolny znak graficzny

Wyrażenie regularne

Do opisu wyrażenia można wykorzystać zdefiniowane w bibliotece klasy

lower	dowolny mały znak (\l)
print	dowolny znak drukowalny
punct	dowolny znak interpunkcyjny
s	dowolny biały znak (\s)
space	jak wyżej
upper	dowolna duża litera (\u)
w	dowolna litera, cyfra lub znak _ (\w)
xdigit	dowolna cyfra szesnastkowa

Wyrażenie regularne

- Numer telefonu : +48 555-444-333

  `\+48 [5-9]{1}[0-9]{2}-[0-9]{3}-[0-9]{3}`
- Adres email: nazwa@serwer.pl

  `[a-z]+@[a-z]+\.[a-z]{2,4}`
- Adres email:
  Imie.Nazwisko@server.com

  `[A-Z]{1}[a-z]*\.[A-Z]{1}[a-z]*@[a-z]+\.[a-z]{2,4}`
- Nip 777-777-77-77

  `[0-9]{3}-[0-9]{3}-[0-9]{2}-[0-9]{2}`
- W kodzie C++ należy pamiętać że znak '\' należy poprzedzić '\'

Wyrażenie regularne

- **Inicjacja biblioteki regex**

```
#include <regex>
```

- **Zadeklarowanie obiektu regex**

```
regex szablon(”[A-Za-z]{5-10}”);

string regula = ”[A-Za-z]{5-10}”;
regex szablon(regula);

regex szablon;
szablon=regula;  //try…catch
```

Wyrażenie regularne

- Sprawdzenie czy ciąg tekstowy jest zgodny z ustalonym wzorcem

```
bool wynik=regex_match(tekst,szablon);
```

- Jeżeli tekst jest zgodny z szablonem zwracana jest wartość true
- Jeżeli tekst nie spełnia kryteriów szablonu funkcja zwraca wartość false

Przykład

```
28  int main()
29  {
30      email kontakt(0);
```

C:\Windows\system32\cmd.exe

```
Podaj email: adam.kowalski@poczta.info
Podano email niezgodny z wybranym standardem!
Podaj email: a.kowalski@poczta.info
Prawidłowy format adresu email!

Press any key to continue . . .
```

```
42      } while (true);
43  }
24      {
25          return regex_match(tekst,szablon);
26      }
31  >   do ···
42      } while (true);
43  }
```

Wyrażenie regularne

- Wyszukiwanie określonego zestawu znaków w analizowanym tekście
- Wzorzec wyszukiwania może być wyrażeniem złożonym
- Funkcja wyszukiwania może analizować cały wzorzec, także zwracać informacje o spełnieniu warunku każdego ze składników wzorca

Wyrażenie regularne

- **Podstawowa postać**

```
bool test =
regex_search(tekst,wzorzec);
```

- **Rozszerzona**

```
smatch wynik;
bool test =
regex_search(tekst,wynik,wzorzec);
wynik[n]; //0-n gdzie n jest
      liczbą składników wzorca,
zwraca informację o pierwszym
spełniającym warunek wzorca
```

Wyrażenie regularne

- **Wyrażenie d wyszukiwania można zapisać zgodnie z zasadami omówionymi wcześniej**

```
string war = "(tekst1)|(tekst2)";
regex warunek(war);
```

- **Można zbudować go z warunków cząstkowych**

```
string war1 = "(?=.*[a-b])";
string war2 = "(?=.*[0-9])";
string warA = war1 + war2;
string warB = war1 + "|" + war2;
regex warunek(warA);
```

Przykład 1

```
1  #include <iostream>
2  #include <regex>
3  using namespace std;
4  int main()
5  {
```

C:\Windows\system32\cmd.exe

```
Podaj tekst do analizy: Ola ma psa, a Ela ma kota. Ala nie ma pupila.
Wynik : 4
Ola ma psa, a Ela ma kota. Ala nie ma pupila. : Ola
Wynik(0): Ola
Wynik(1):
Wynik(2): Ola
Wynik(3):

Press any key to continue . . .
```

```
16                  cout << "Wynik(" << i << "): " << wynik[i] << endl;
17          }
18          else
19              cout << "Nie znaleziono wzorców." << endl;
20  }
```

Przykład 2

```
#include <iostream>
#include <regex>
using namespace std;
int main()
{
    string tekst;
    smatch wynik;
    if( regex_search( tekst, wynik, wzorzec ) )
        {
            cout << "Wynik : " << wynik.size() << endl;
            cout << tekst << " : " << wynik[0] << endl;
            for(int i=0;i<wynik.size();i++)
                cout << "Wynik(" << i << "): " << wynik[i] << endl;
        }
        else
            cout << "Nie znaleziono wzorców." << endl;
}
```

```
C:\Windows\system32\cmd.exe
Podaj tekst do analizy: Ola ma kota, a Ela ma psa. Ala nie ma pupila.
Wynik : 1
Ola ma kota, a Ela ma psa. Ala nie ma pupila. :
Wynik(0):

Press any key to continue . . .
```

Przykład 3

```cpp
#include <iostream>
#include <regex>
using namespace std;
int main()
{
    string tekst;
    smatch wynik;
    string war1 = "(?=[Ala])";
    string war2 = "(?=[Ola])";
    string war3 = "(?=[Ela])";
    string war = war1 + "|" + war2 + "|" + war3;
    regex wzorzec(war);
    cout << "Podaj tekst do analizy: ";
    getline(cin,tekst);
    if( regex_search( tekst, wynik, wzorzec ) )
        {
            cout << "Wynik : " << wynik.size() << endl;
            cout << tekst << " : " << wynik[0] << endl;
            for(int i=0;i<wynik.size();i++)
                cout << "Wynik(" << i << "): " << wynik[i] << endl;
        }
        else
            cout << "Nie znaleziono wzorców." << endl;
}
```

```
C:\Windows\system32\cmd.exe
Podaj tekst do analizy: Ala ma psa.
Wynik : 1
Ala ma psa. :
Wynik(0):

Press any key to continue . . .
```

07:46

Przykład 4

```
12      bool test()
13      {
14          bool wynik = regex_search(pass,wyrazenie);
15              if(!wynik)
16                  cout << endl << "Hasło nie spełnia wymagań" << endl;
17          return wynik;
18      }
28          war1="(?=.*[-_!@$])";
```

```
42                  if((int)c==13) break;
```

C:\Windows\system32\cmd.exe

```
Hasło powinnoi zawierać: a-zA-Z0-9-_!@$ : od 6 do 12 znaków
**************
Hasło nie spełnia wymagań
*******
Hasło nie spełnia wymagań
************
HASŁO: Alfa34_!$jfh

Press any key to continue . . .
```

```
60      cout << "Hasło powinnoi zawierać:
61      haslo.setPass();
62      cout << "HASŁO: " << haslo.show()
63  }
```

```
53      {
54          return pass;
55      }
```