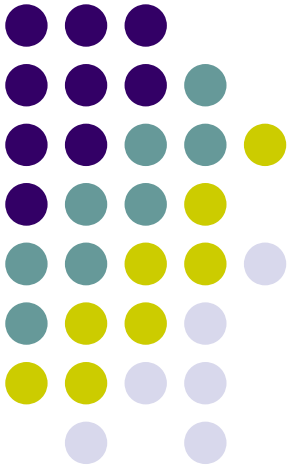
Średniozaawansowane programowanie w C++

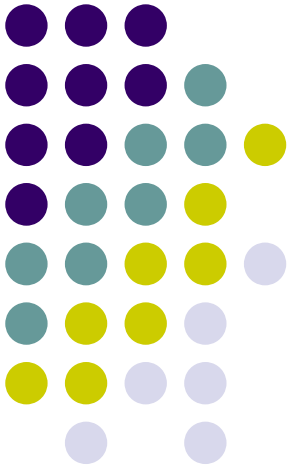
Wykład #7

2 grudnia 2020 r.

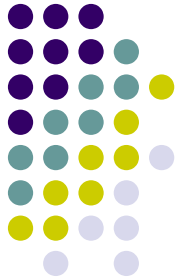


Szablony

Wprowadzenie



Wady i zalety



Zalety

- Uniwersalne zastosowanie kodu (dla dowolnych klas spełniających określone kryteria)
- Wysoka wydajność

Wady

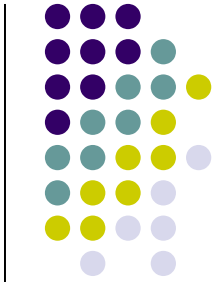
- Wielokrotna kompilacja kodu
- Kod jest kompilowany dopiero w momencie użycia
- Niezrozumiałe, rozwlekłe komunikaty o błędach

Organizacja kodu

Plik stos.hpp – definicja klasy

```
#ifndef _stos_hpp_
#define _stos_hpp_

template <typename T> class Stos
{
        public:
                Stos ();
                void poloz (const T&);
                T zdejmij ();
                unsigned rozmiar () const;
                unsigned zajete () const;
        private:
                // (…) składowe prywatne
};

#include "stos_impl.hpp"

#endif
```

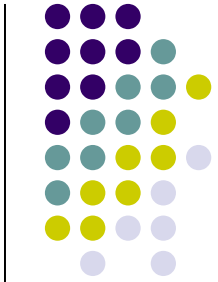
Organizacja kodu



Plik stos_impl.hpp – implementacja metod zależnych od T

```
#ifndef _stos_impl_hpp_
#define _stos_impl_hpp_

template <typename T> void Stos<T>::poloz (const T&)
{
        /* definicja metody */
}

template <typename T> T Stos<T>::zdejmij ()
{
        /* definicja metody */
}

// (…) inne metody szablonu

#endif
```

Organizacja kodu



Plik stos_cpp – implementacja metod niezależnych od T

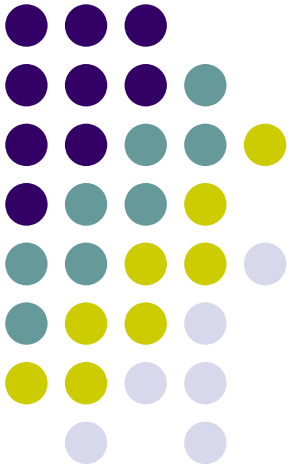
```
#include _stos_hpp_

unsigned Stos::rozmiar () const
{
        /* definicja metody */
}

unsigned Stos::zajete () const
{
        /* definicja metody */
}

// (…) inne metody szablonu
```

Szablony

Szablony funkcji

Szablony klas

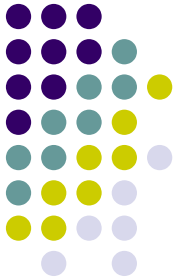
Szablony zmiennych (C++14)



Szablony funkcji

```
template <typename T> void zamien (T &a, T &b)
{
        T tmp = a;
        a = b;
        b = tmp;
}

template <typename T> T tabs (const T &x)
{
        if (x < 0)
                return -x;
        else
                return x;
}

// Użycie funkcji:
int x = 3, y = 7;
float z = -3.14159;
zamien (x, y);  // automatyczna konkretyzacja zamien<int> (x, y);
float modz = tabs (z); // automatyczna konkretyzacja tabs<float> (z);
```

Szablony klas



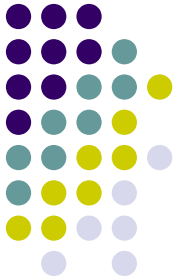
```
template <typename T, unsigned n> class Stos
{
    public:
        Stos ();
        Void poloz (const T &tt);
        T zdejmij ();
    private:
        T stos_ [n];
        T *element_;
};
```

UWAGA! Parametrem szablonu może być nie tylko typ/klasa, ale także konkretny obiekt wskazanego typu!

Specjalizacja szablonów

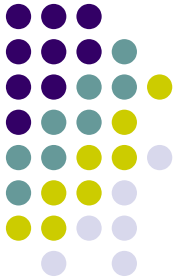
```
template <class X> void moja_funkcja (const X &xx)
{
        // (…) definicja funkcji
}

// Specjalizacja szablonu dla wskaźników
template <class X> void moja_funkcja<X*> (const X xx)
{
        // (…) definicja funkcji
}

// Specjalizacja szablonu dla typu int
template <> void moja_funkcja<int> (const int &xx)
{
        // (…) definicja funkcji
}
```

Szablony zmiennych



```
// Przykład #1
template <class T>
constexpr T pi = T(3.14159265358979323851);  // variable template

template <class T>
T circular_area(T r) // function template
{
    return pi<T> * r * r;
}

// Przykład #2
template <int N> const int ctSquare = N*N;
std::cout << ctSquare<7> << std::endl;
```

Szablony zmiennych (2)

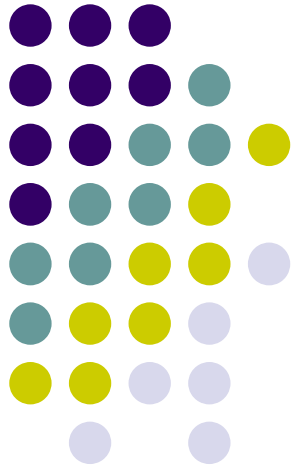


```
// Przykład #3
template <typename T>
        T n = T(5);

int main()
{
    n<int> = 10;
    std::cout << n<int> << " ";    // 10
    std::cout << n<double> << " "; // 5
}
```

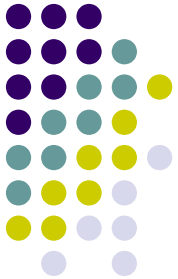
Szablony

Traits

if constexpr ()



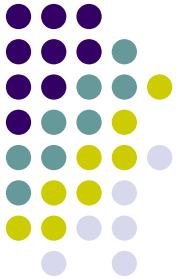
Standardowe trejty



```
#include <limits>

template <class T> T* element_maksymalny (T *poczatek, T *koniec)
{
        T max = std::numeric_limits<T>::min();  // trait :)
        T *pmax;
        for (T *it = poczatek; it != koniec; ++it)
            if (*it > max)
            {
                max = *it;
                pmax = it;
            }
        return pmax;
}
```

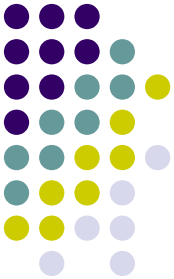
Więcej zapierających dech w piersiach traitów:
www.cplusplus.com/reference/std/limits/numeric_limits/
www.boost.org/doc/libs/release/libs/type_traits/

Własne traity (1)



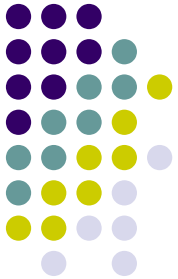
```
// Klasy do rozróżniania sposobu głaskania
struct sposob_glaskania {};

struct glaszcz_raczka : public sposob_glaskania {};

struct glaszcz_szczotka : public sposob_glaskania {};

template <typename Zwierzatko> struct glaskanie_trait {
        typedef typename Zwierzatko::jak_glaskac jak_glaskac;
};

// Zwierzątka do głaskania
class Kroliczek {
    public:
        typedef glaszcz_raczka jak_glaskac;
        /* (…) */
};

class Krowka {
    public:
        typedef glaszcz_szczotka jak_glaskac;
        /* (…) */
};
```

Własne traity (2)



```
// Funkcje wyspecjalizowane
template <typename Zwierzatko>
void glaszcz_zwierzaczka_t (Zwierzatko &z, glaszcz_raczka)
{
        // (…) głaszcze rączką
}

template <typename Zwierzatko>
void glaszcz_zwierzaczka_t (Zwierzatko &z, glaszcz_szczotka)
{
        // (…) głaszcze szczotką
}

// Uniwersalna funkcja wołana przez użytkownika
template <typename Zwierzatko>
void glaszcz_zwierzaczka (Zwierzatko &z)
{
        // Woła funkcję wyspecjalizowaną
        glaszcz_zwierzaczka_t (z,
                typename glaskanie_trait<Zwierzatko>::jak_glaskac ());
}
```

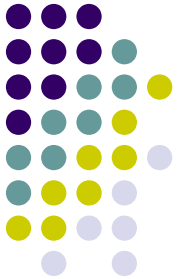
if constexpr ()



```
template <class T>
        constexpr T absolute(T arg)
        {
           return arg < 0 ? -arg : arg;
        }

template <class T>
        constexpr auto precision_threshold = T(0.000001);

template <class T>
        constexpr bool close_enough(T a, T b) {
           if constexpr (is_floating_point_v<T>)
              return absolute(a - b) < precision_threshold<T>;
           else
              return a == b;
}
```

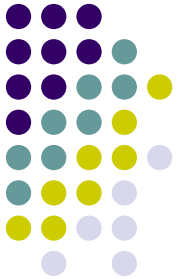
A w ogóle co to jest to **constexpr**...?!?
https://en.cppreference.com/w/cpp/language/constexpr

if constexpr ()



```
struct S
{
    int n;
    std::string s;
    float d;
};

template <std::size_t I>
  auto& get(S& s)
  {
    if constexpr (I == 0)
        return s.n;
    else if constexpr (I == 1)
        return s.s;
    else if constexpr (I == 2)
        return s.d;
  }

S obj { 0, "hello", 10.0f };
std::cout << get<0>(obj) << ", " << get<1>(obj) << "\n";
```

Ewaluacja wyrażeń podczas kompilacji



```
constexpr int factorial (int n) {
    return n <= 1 ? 1 : (n * factorial (n - 1));
}

constexpr int fibonacci (unsigned n) {
    return n <= 1 ? n : fibonacci (n - 1) + fibonacci (n - 2);
}

// Kompilator zastąpi wywołanie funkcji konkretnymi wartościami!
std::cout << factorial (5)  << '\n';
std::cout << fibonacci (10) << '\n';
```

Programowanie jest fantastyczne!!!