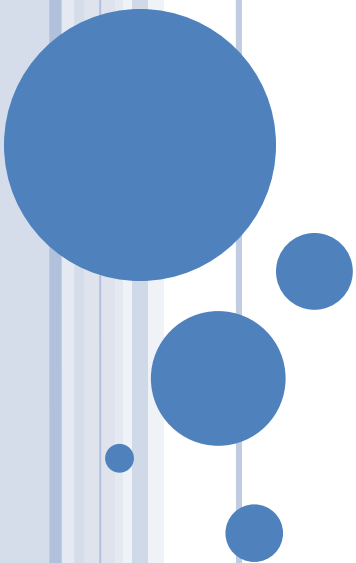




KURS JĘZYKA C++ – WYKŁAD 9 (25.04.2018)

Szablony

SPIS TREŚCI

- Szablony w programowaniu
- Definicja szablonu
- Parametry w szablonie
- Konkretyzowanie szablonu
- Przeciążanie szablonów funkcji
- Dopasowanie i generowanie funkcji szablonowych
- Szablony funkcji z biblioteki standardowej
- Użycie argumentów wzorca do specyfikowania strategii
- Domyślne parametry szablonu
- Specjalizacja szablonów klas



SZABLONY

- Szablon (inaczej wzorzec) to przezroczysta dla programu konstrukcja językowa, na podstawie której kompilator jest w stanie wygenerować zbiór podobnych funkcji lub podobnych klas.
- Szablon zastępuje w programowaniu żmudne operacje kopiowania, wklejania i drobne modyfikacji kodu.
- Szablony są sparametryzowane przede wszystkim za pomocą typów (ale także pewnych wartości).
- Prawie wszystkie klasy i funkcje z biblioteki standardowej są szablonami.



SZABLONY

- Szablony bezpośrednio wspomagają programowanie uogólnione, czyli programowanie z użyciem typów jako parametrów.
- Za pomocą szablonów można łatwo reprezentować i łączyć ze sobą ogólne koncepcje programistyczne (algorytmy oraz struktury danych).
- Szablon zależy tylko od tych właściwości typów swoich parametrów, których rzeczywiście używa.
- Argumentami szablonów mogą być i często są typy wbudowane.
- Kompozycje składane z szablonów są bezpieczne pod względem typów, ale niestety wymagań szablonu co do jego argumentów nie da się prosto i bezpośrednio wyrazić w kodzie.

SZABLONY

- Możemy definiować szablony funkcji i szablony klas.
- Szablony definiuje się umieszczając przed definicją funkcji lub klasy frazę `template` z listą parametrów w nawiasach kątowych.

- Przykład:

```
template <typename T>
T maksimum (const T &a, const T &b)
{
    return a < b ? b : a;
}
```

Na podstawie tej definicji kompilator umie wygenerować funkcję `maksimum()` dla obiektów różnych typów (dla których zdefiniowano operator porównywania `operator<`).



OKREŚLENIE TYPU `TYPENAME`

- Słowo `typename` wskazuje, że następujący po nim identyfikator jest nazwą typu.

- Przykład:

```
template <typename T>
class MyClass {
    typename T::SubType *ptr;
    // ...
};
```

W przykładzie tym `ptr` jest wskaźnikiem na obiekt typu `T::SubType` (a nie iloczynem składowej statycznej `T::SubType` przez `ptr`).



PARAMETRY SZABLONU

- Parametr szablonu może być:
 - typem (oznacza się go jako class lub typename),
 - wartością porządkową (może to być char, int itp., oraz wskaźnik),
 - wartością wcześniejszego typu będącego parametrem szablonu.
- Szablon może mieć wiele parametrów.
- Przykłady:

```
template <class T, int rozm>
    class Bufor {...};
template <class T, T wart>
    class Schowek {...};
```



DEFINIOWANIE SZABLONU

- Szablon definiuje się w pliku nagłówkowym, gdyż kompilator musi znać jego definicję, aby na jej podstawie wygenerować funkcję lub klasę szablonową.
- Szablon funkcji lub klasy może się pojawiać wielokrotnie w pliku (poprzez włączenie pliku nagłówkowego) i nie spowoduje błędu (tak jak definicja funkcji wbudowanej).



DEFINIOWANIE SZABLONU

- Składowe szablonu klasy definiuje się tak samo jak dla zwykłej klasy.
- Funkcje składowe szablonu można definiować poza klasą (ale tak by kompilator widział te definicje).
- Składowe szablonu klasy same są szablonami i są sparametryzowane parametrami swoich szablonów klas.



DEFINIOWANIE SZABLONU

Przykład szablonu klasy:

```
template <typename T, int rozm>
class stos {
    T tab[rozm];
    int ile;
public:
    void wstaw (const T &x) throw(std::out_of_range);
    T zdejmij () throw(std::out_of_range);
    int rozmiar () const { return ile; }
};

template <typename T, int rozm>
void stos::wstaw (const T &x) throw(std::out_of_range) {
    if (ile>=rozm) throw std::out_of_range("przepełnienie stosu");
    tab[ile++] = x;
}

template <typename T, int rozm>
T stos::zdejmij () throw(std::out_of_range) {
    if (ile<=0) throw std::out_of_range("wyczerpanie stosu");
    return tab[--ile];
}
```



KONKRETYZOWANIE SZABLONU

- Funkcja szablonowa to funkcja wygenerowana przez kompilator na podstawie szablonu funkcji.
- Kompilator wygeneruje funkcję szablonową, gdy napotka jej wywołanie lub pobranie adresu funkcji.
- Kompilator sprecyzuje typ funkcji szablonowej na podstawie jej argumentów wywołania (typ rezultatu jest nieistotny).
- Można też jawnie wskazać typ funkcji szablonowej.

- Przykłady:

```
maksimum(x, 192); // x jest typu int
maksimum<double>(2.72, x); // x jest typu int
char (*fun)(const char&, const char&)
    = maksimum;
maksimum(x, 'x'); // błąd - maksimum(int, char);
```



SZABLON FUNKCJI

- Parametrem szablonu funkcji jest przede wszystkim nazwa typu (wartości zwykle przekazuje się jako argumenty do funkcji).
- Kompilator wygeneruje funkcję szablonową, gdy napotka jej wywołanie albo gdy w programie używamy adresu takiej funkcji.
- W szablonie funkcji może wystąpić deklaratorem `inline`.



DOPASOWANIE I GENEROWANIE FUNKCJI SZABLONOWYCH

- Dopasowanie funkcji do szablonu następuje poprzez typy argumentów wywołania funkcji (typ rezultatu nie ma znaczenia).
- Jawną specyfikację stosuje się często w odniesieniu do typu wyniku.

Przykład:

```
template <class T, class U>
T impl_cast (U u)
    { return u; }
```

...

```
void fun (int i)
{
    impl_cast(i); // błąd - nieznane T
    impl_cast<double>(i); // T to double
    impl_cast<char,double>(i); // ok
    impl_cast<char*,int>(i); // błąd - rzutowanie
                             // int na char*
}
```



PRZECIĄŻANIE SZABLONU FUNKCJI

- Szablon funkcji można przeciążać (podobnie jak samą funkcję).
- Można zadeklarować kilka szablonów funkcji o takiej samej nazwie, a także kombinację szablonów i zwykłych funkcji.
- Reguły rozstrzygania przeciążenia w obecności szablonów funkcji są uogólnieniem zwykłych reguł rozstrzygania przeciążenia funkcji:
 - najpierw dla każdego szablonu znajduje się specjalizację, która jest najlepsza dla ciągu argumentów funkcji;
 - następnie stosuje się do tych specjalizacji i wszystkich zwykłych funkcji normalne reguły rozstrzygania przeciążenia.



UŻYCIE ARGUMENTÓW SZABLONU DO SPECYFIKOWANIA STRATEGII

- Problem: sortowanie łańcuchów względem różnych kryteriów porównywania.
- Rozwiązanie:

```
template <class T, class C>
int porownaj (const Napis<T> &a, const Napis<T> &b) {
    for (int i=0; i<a.len()&&i<b.len(); i++)
        if (!C::eq(a[i],b[i]))
            return C::lt(a[i],b[i])?-1:1;
    return a.len() - b.len();
}

template <class T>
class por { public:
    static bool eq (T a, T b) { return a==b; }
    static bool lt (T a, T b) { return a<b; }
};

...
void f (Napis<char> x, Napis<char> y) {
    porownaj<char, por<char>>(x,y);
    ...
}
```



DOMYŚLNE PARAMETRY SZABLONU

- Szablon może mieć parametry domyślne (podobnie jak funkcja może mieć argumenty domyślne). Przykład:

```
template <typename T, typename C=por<T>>  
int porownaj (const Napis<T> &a, const Napis<T> &b) {...}
```
- Technika dostarczania strategii jako argumentu wzorca jest powszechnie wykorzystywana w bibliotece standardowej.
- Parametry wzorca służące do dostarczania strategii nazywa się trejtami (ang. *traits*). Przykładami trejtów są iteratory i alokatory.



SZABLONY KLAS

- Przykład szablonu klasy:

```
template <typename T>
class schowek {
    T tajne;
public:
    schowek (const T &t) : tajne(t) {}
    schowek & operator= (const schowek s)
    {
        if (&s!=this) tajne = s.tajne;
        return *this; }
    T wartosc () { return tajne; }
};
```

- Przykłady klas szablonych:

```
schowek<int> pon(3), wto(7);
schowek<char> sro('s');
schowek<string> czw("czwartek");
```



SZABLONY KLAS

- Szablon klasy a klasy szablonowe.
- Parametry formalne szablonu (te w nawiasach kątowych) a parametry aktualne (konkretny typ dla klasy szablonowej).
- Nazwa szablonu klasy musi być unikalna.
- Szablon klasy musi mieć zasięg globalny – nie wolno zagnieżdżać definicji jednego szablonu w drugim.



SZABLONY KLAS

- Uwaga na składniki statyczne w szablonie.
- Instrukcje `typedef` i `enum` w szablonie klasy.
- Przyjaźń a szablony klas...
 - przyjaciel ogólny
 - przyjaciel szablony
 - zaprzyjaźnione operatory we/wy
- Dziedziczenie a szablony klas...



SPECJALIZACJA SZABLONÓW KLAS

- Alternatywne definicje wzorca nazywa się specjalizacjami.

- Przykład:

```
template <class T>
    class Wektor {...};
template <class T>
    class Wektor<T*> {...};
template <const char *>
    class Wektor<const char*> {...};
template <>
    class Wektor<void*> {...};
```



SPECJALIZACJA SZABLONÓW KLAS

- Tworząc specjalizację szablonu jakiejś klasy, możemy w niej zdefiniować inne pola i metody niż w szablonie ogólnym.
- Mając wzorzec klasy, można w nim wyspecjalizować tylko wybrane funkcje składowe, zamiast tworzyć specjalizację całej klasy.
- W rozstrzyganiu przeciążenia preferuje się wersję najbardziej specjalizowaną.



ALIASY SZABLONÓW

- Możliwe jest definiowanie aliasów dla szablonów, nawet z niezdefiniowanymi parametrami szablونowymi.
- Przykład:

```
template <typename first,  
        typename second, int third>  
    class SomeType;  
template <typename second>  
    using TypedefName =  
        SomeType<OtherType, second, 5>;
```

